UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Desarrollo de un juego de estrategia por turnos

Autor/es

DIEGO RUIZ LERA

Director/es

JÓNATAN HERAS VICENTE

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2018-19

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en ingeniería informática

Desarrollo de un juego de estrategia por turnos

Realizado por:

Diego Ruiz Lera

Tutelado por:

Jónathan Heras Vicente.

Logroño, junio, 2019

Resumen

Desde lo más remoto de los tiempos, los humanos han intentado invertir su tiempo libre en acciones que les resulten agradables o relajantes. Entre ellas se encuentran los juegos como el ajedrez o el parchís. En la actualidad la existencia de las nuevas tecnologías nos han permitido trasladar los juegos a ordenadores y otros dispositivos creando con ello los videojuegos.

Los videojuegos, queramos o no, son una industria que mueve mucho dinero. Todos los días millones de personas juegan. Seguramente cualquier persona con un dispositivo electrónico habrá jugado a alguno a lo largo de su vida. En definitiva, es un tipo de software muy implantado en la sociedad.

A pesar de ello, por alguna razón mucha gente no los ve como un negocio o como proyectos software “serios”. Esto contrasta con el volumen económico movido por los videojuegos.

Este Trabajo Fin de Grado tiene como objetivo demostrar, que el empleo de técnicas y sistemas de gestión usadas en cualquier desarrollo software son válidas para el desarrollo de un videojuego. Para ello se usarán herramientas y técnicas *AGILE* además del desarrollo de prototipos que serán testeados a lo largo del proyecto para crear un videojuego de estrategia por turnos.

Abstract

Since the beginning of time, humans have used their spare time to engage in entertaining and relaxing activities such as playing games. Among these games we can find chess or cards. In the last few years, and thanks to the new technology advancements, we have been able to transfer all these games to computers and other devices. And with this action, we have created videogames.

Whether we like it or not, the videogame industry is a multimillionaire one as everyday millions of people play videogames. It is very likely that any person that has a device, would have played at least one game in their lifetime. Ultimately, videogames are a type of software with a lot of presence in our current society.

Despite all these facts and numbers, for some reason, some people do not see the videogame industry as a business or creating a videogame as a serious software project. This presents a stark contrast with the economic impact of the videogames industry.

The objective of this final degree project is to demonstrate that the usage of techniques and management systems used in any software development are valid for the development of a videogame. To illustrate this, I have created a turn-based strategy videogame using AGILE tools and techniques as well as prototypes that have been tested at different stages of the development of the software project.

Índice

1. Introducción

En este apartado se dará una explicación sobre distintos conceptos relacionados con los videojuegos que son necesarios para entender el contenido y propósito de este TFG.

1.1 Antecedentes

La creación de videojuegos es una de las industrias del entretenimiento que más ha crecido en la última década en el mundo [1]. En enero de 2019 se publicó un informe económico que habla sobre el crecimiento de la industria del videojuego en España [2]. En él se indicaba que la industria del videojuego está en el segundo puesto (detrás de la literatura) en industrias culturales que más facturan en el país y se estima que supere a la literatura en el año 2023, ver imagen 1. Con esto se quiere recalcar que la industria del videojuego en España puede convertirse en un motor económico-cultural de gran importancia y debe tenerse en cuenta como posible sector de trabajo para ingenieros informáticos.

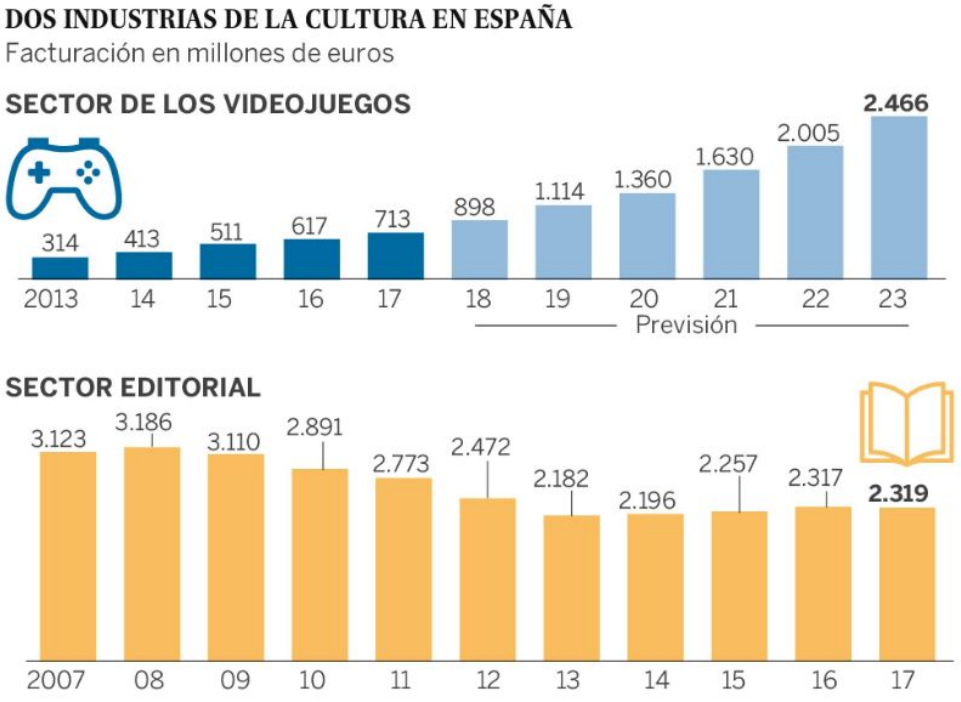


Imagen 1: *previsiones económicas de la industria del videojuego [2]*

Dada la importancia que los videojuegos pueden llegar a tener, puede ser interesante aprender sobre herramientas y técnicas que se usan en esta rama del desarrollo software.

A la hora de abordar la creación de un videojuego hay que ser conscientes de que están compuestos por distintos apartados técnicos: diseño de niveles, guión (si tiene historia), calidad gráfica, banda sonora... Uno de los componentes más importantes y que diferencia a los videojuegos del resto de artes (algunos denominan a los videojuegos como octavo arte) son las mecánicas.

¿Que es una mecánica?

Dependiendo las fuentes consultadas las mecánicas se pueden interpretar de distintas formas [3].

En primer lugar hay que aclarar los términos *game state* y *gameplay* para entender correctamente qué es una mecánica. Las siguientes definiciones también pueden tener interpretaciones, pero en este documento se entenderán estos elementos con las definiciones que se dan a continuación.

- *Game state*: es una "instantánea" de un juego. En él se almacenan todos los objetos del juego y sus valores en un momento dado. Cualquier pequeña modificación sobre los valores o sobre la disposición de los objetos altera el *game state [4].*

- *Gameplay*: se puede traducir como "jugabilidad" y esta se determinará según las acciones que puede realizar un jugador en el entorno del juego. Por ejemplo, hacer saltar un personaje tras pulsar una tecla determinada [5].

A partir de estas dos nociones existen mayormente 2 posibles definiciones de mecánica [6]:

1) Las mecánicas son las acciones de las que dispone el jugador para actuar sobre el *game state.* Que prácticamente es lo mismo que es decir que el *gameplay* es sinónimo de las mecánicas.

Esta definición a priori correcta deja muchas cosas que suceden en los videojuegos sin denominación. Una (Inteligencia Artificial) IA no es el jugador pero también cambia el *game state*. Por lo tanto una definición alternativa de mecánica es:

2) Las mecánicas son todos los métodos invocados por agentes (no hace falta que sea el jugador) diseñados para interaccionar con el *game state*.

La segunda definición es más completa, por lo que será la usada en este documento como válida.

Pero a niveles prácticos ¿cómo podemos observar una mecánica? Una mecánica realmente es una regla que se cumple cuando sucede algo y modifica el *game state*. Por lo tanto, las mecánicas pueden ser cosas básicas. Por ejemplo en "Super Mario Bros" una mecánica es saltar (cuando el jugador pulsa el botón de saltar el avatar salta) y otra es eliminar un enemigo (si un enemigo recibe suficiente daño este es eliminado). Estas se pueden unir y generar otras mecánicas como: "si Mario salta sobre un enemigo éste recibe daños y es eliminado", ver imagen 2.

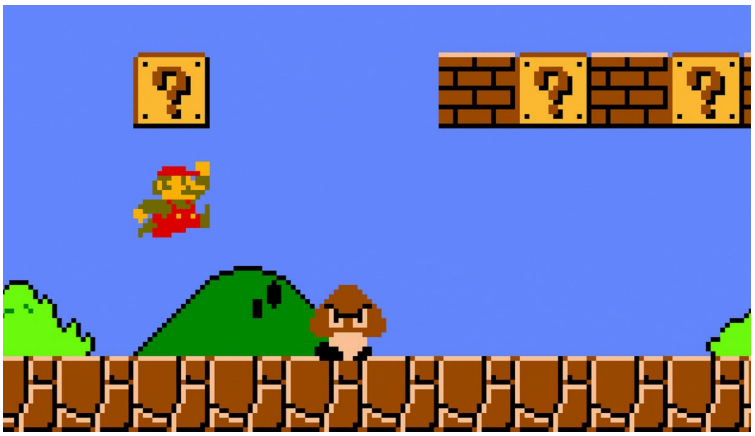


Imagen 2: *captura[1] del juego Super Mario Bros (1985)*

Las mecánicas son muy importantes, lo suficiente como para existir videojuegos completamente basados en ellas (por ejemplo el pong o el tetris). Pero en la actualidad las mecánicas se deben mezclar con los otros elementos de un videojuego (narrativa, banda sonora...). Cuando esto no se cumple se pueden producir lo que algunos denominan "disonancia ludonarrativa" [7]. Esto sucede cuando las mecánicas de un juego no acompañan al resto de elementos. Por ejemplo si queremos contar una historia sobre el significado de la paz (narrativa) lo más normal no sería hacer que el personaje se dedicase a eliminar miles de entes (humanos o no) a balazos de forma frenética. Posiblemente prefiramos unas mecánicas más pausadas que nos den mucho tiempo entre acción y acción para digerir la historia que se nos quiere contar. Como en las películas o libros dependiendo de lo que se quiera narrar deberán usarse marcos donde desarrollar la acción. En los videojuegos estos marcos suelen ser denominados géneros.

¿Que es un género en videojuegos?

Podríamos discutir si las mecánicas determinan el género de videojuego o si es el género el que delimita las mecánicas. Una anécdota respecto a este tipo de debate es la siguiente:

En el año 1874 en París al hilo de una exposición artística de artistas independientes el crítico de arte Louis Leroy definió de forma despectiva los cuadros que allí se encontraban como "impresionistas". Años más tarde todo ese conglomerado de cuadros de trazos poco definidos pasó a ser un movimiento artístico con ese nombre.

Algo similar sucedió de forma menos épica con los distintos géneros donde poder clasificar los distintos videojuegos existentes. En 1984 Chris Crawford en su libro *"The art of computer game design" [8]* genera la primera clasificación de los videojuegos, esta se hizo principalmente según las mecánicas.

Las mecánicas como es obvio no surgieron todas a la vez y luego se diseñaron los géneros para acotarlas. Las mecánicas se fueron creando gracias a avances técnicos en la industria y a gente que se le ocurrieron ideas de cómo añadir distintas características al juego para hacerlo diferenciador en el mercado.

[1] Imagen obtenida de https://www.t13.cl/noticia/tendencias/super-mario-bros.-cumple-33-anos

Actualmente la gestación de un nuevo género se produce cuando alguien lanza un juego (o un modo de juego) que implementa unas nuevas reglas (mecánicas) que hacen de él una experiencia distinta. Si tiene éxito, otros videojuegos comenzarán a implementar estas nuevas mecánicas. Si muchos videojuegos implementan estas nuevas mecánicas y son bastante diferenciadoras, el público (periodistas mayoritariamente) acaban acuñando un nombre a los videojuegos que las implementen y así se genera un nuevo género.

En la actualidad esa lista de Chris Crawford se podría considerar obsoleta, pero constituyó las bases para las actuales clasificaciones que están continuamente creciendo. Por ejemplo en los últimos 7 años hemos visto aparecer el género "*Battle royale*" desde su humilde inicio como un modo de juego online del videojuego Minecraft, hasta los actuales éxitos comerciales como PUG (*PlayerUnknown´s Battlegrounds*) o Fornite, ver imagen 3.

Top premium PC and console games by revenue, 2018

Rank	Title	Publisher	Genre	Revenue
1	*PlayerUnknown's Battlegrounds*[1]	Bluehole	Shooter	$1,035M
2	*FIFA 18*	Electronic Arts	Sports	$830M
3	*Red Dead Redemption 2*	Take-Two Interactive	Action-Adventure	$732M

Top free-to-play games by revenue, 2018

Rank	Title	Publisher	Genre	Revenue
1	*Fortnite*	Epic Games	Shooter	$2.4B
2	*Honour of Kings*	Tencent	MOBA	$2.1B
3	*Dungeon Fighter Online*	Nexon	RPG	$1.5B

Imagen 3: *capturas de los top ventas 2018 donde se pueden ver los ingresos del PUG y Fornite [9]*

Existen muchos géneros y subgéneros como son *First Person Shooter* (FPS) , *Battle royale*, survival, simulador… Esta clasificación es tan usada que periodistas del medio y jugadores suelen usarla a la hora de explicar un juego. Estos nombres no dicen casi nada sobre la narrativa (aunque como he dicho antes sería raro ver una historia sobre la paz mundial en un FPS) pero si se explica cómo se controlará el juego y que se puede esperar de él a la hora de jugar (su ritmo de juego, que cosas se podrán hacer y cuáles no, las horas que seguramente cueste acabarlo...).

En ocasiones se usan videojuegos que marcaron historia como nombre de género. Por ejemplo "*metroidvania*" es un subgénero de videojuegos que te está indicando que el juego tendrá una estética y unas mecánicas muy parecidos a los videojuegos de las sagas Metroid y Castlevania.

Como se podrá deducir, dependiendo de que queremos mostrar al jugador usaremos un género de juego u otro. Si queremos centrarnos en la narrativa y que la experiencia sea más pausada usaremos géneros como "*walking simulator*" o "*Point and click"* que usan pocas mecánicas y suelen ser pausados dando tiempo a que cale el mensaje en el usuario, mientras que otros géneros como "estrategia por turnos" se centran mucho en el apartado mecánico reduciendo la importancia del resto de apartados y abrumar al jugador con muchas pequeñas mecánicas entrelazadas.

En este trabajo se implementarán distintas mecánicas de un juego de estrategia por turnos, por lo que se pasa a explicar de forma más detallada este género.

¿Que es un juego de estrategia por turnos?

Un videojuego de estrategia por turnos se basa en que el usuario disponga de unas fichas que se desplazarán por un tablero (formado por celdas) con el objetivo de vencer a las fichas del otro jugador, o de llegar a una posición determinada del tablero. Los jugadores moverán todas las unidades de forma asíncrona, por turnos. Como se puede ver las mecánicas básicas del género son muy parecidas a las reglas (mecánicas) del ajedrez.

Existen unas características "generales" (aparte de las mecánicas) en las que se engloban esta clase de videojuegos. Dichas características suelen exigir al jugador que medite sus movimientos antes de actuar, el apartado gráfico no suele ser prioritario y la narrativa no suele tener gran importancia. Estos videojuegos centran la experiencia de juego en las mecánicas. En las imágenes 4 y 5 pueden verse capturas de videojuegos de este género donde se ve que el apartado gráfico no es muy alto.

Imagen 4: *captura del juego Fire Emblem Echoes: Shadows of Valentia*

Imagen 5: *captura del juego Panzer Tactics HD*

La cantidad de mecánicas existentes en estos videojuegos es inmensa y cada juego implementa unas u otras para que exista una coherencia entre las mismas.

Aunque esta clase de videojuegos no suelen ser muy populares, algunos han tenido gran éxito comercial y de crítica (como por ejemplo *X-COM enemy unknown* [10]), ver imagen 6. Como puede imaginarse las mecánicas más avanzadas son lo que hace de estos videojuegos grandes títulos.

Imagen 6: *captura del juego X-COM enemy unknown*

Una de las mecánicas que más impresión suscitó en el juego de *X-COM enemy unknown* es la probabilidad de fallo al disparar. En la mayoría de videojuegos cuando a una unidad se le ordena atacar a otra, suele tener la certeza de que el ataque golpeará a la unidad adversaria. En *X-COM* eso no sucede ya que existe una probabilidad de que unidad erre el disparo y se pierda el turno de esa unidad. Esta mecánica no era acompañada por la representación gráfica del juego (imagen 7). Por lo que se generaban situaciones incoherentes para los jugadores. Paradójicamente estos errores hicieron que el juego fuese más exitoso.

Imagen 7: *captura*[2] *de X-COM que muestra a modo de sátira como la mecánica actúa*

1.2 Objetivo

Una vez introducidos los distintos conceptos sobre videojuegos podemos enunciar el objetivo de este trabajo. Como ya se ha indicado, los videojuegos tienen múltiples apartados, abordarlos todos es imposible en un TFG. Por lo tanto, este TFG solo se centrará en las mecánicas y en la interfaz.

- El apartado mecánico de este proyecto consistirá en crear un número suficiente de mecánicas interconectadas entre sí, con el objetivo de crear un *gameplay* profundo e interesante para el jugador.

- Las interfaces que se desarrollaran en este trabajo tendrán como objetivo, hacer que el jugador le resulte cómodo y entendible el uso de las mecánicas de las que dispondrá el jugador. La interfaz debe evitar que el jugador no comprenda lo que ocurre en la partida.

1.3 Motivación

La razón principal para la realización de este proyecto es que me resulta interesante poder realizar un proyecto completo y de envergadura desde la planificación hasta la implementación usando técnicas *AGILE (scrum).*

El que este proyecto sea supervisado, guiado y evaluado también ayuda a que eligiera hacerlo por mi cuenta de forma que pueda usarlo como evaluación personal.

Otra razón es mi falta de experiencia con herramientas de desarrollo de videojuegos, realizar este proyecto puede ser una buena toma de contacto con este tipo de desarrollo.

[2] Imagen obtenida en https://knowyourmeme.com/photos/1078995-x-com del autor " the big bad memebag"

2. Planificación

En esta sección se van a abordar los distintos apartados referentes a la planificación. Estos apartados tratan sobre el alcance del proyecto, cómo se estructura el proyecto y la gestión de los riesgos existentes a lo largo de su ejecución.

2.1 Alcance

Este proyecto tiene como objetivo crear un prototipo de juego de estrategia por turnos que contenga las suficientes mecánicas e interfaces como para que sea jugable.

Al disponer de unas horas limitadas (aproximadamente 300 h) el proyecto se centrará en crear distintas mecánicas que hagan del prototipo un juego de estrategia por turnos (que cumpla las mecánicas básicas de este género) y se le añadirán otras mecánicas para darle características diferenciadoras sobre otros videojuegos del género.

El resto de apartados técnicos (narrativa, banda sonora, gráficos...) se mantendrán al mínimo. Aunque se hará un esfuerzo para hacer que la interfaz sea de calidad para el jugador. Este esfuerzo se traducirá en la obtención de información sobre la interfaz del prototipo. Para obtener esta información se realizaran unas encuestas a lo largo del proyecto a unos voluntarios que harán de testeadores. Con la información obtenida se mejorarán la interfaz y en caso de no poder (las mejoras sean muy costosas o se necesitan más conocimientos gráficos que técnicos) se pondrá por escrito las mejoras que habría que realizar.

Por lo tanto, el prototipo final deberá tener una interfaz útil que ayude al jugador a entender la situación de sus unidades (las fichas que podrá mover el jugador) en el campo de batalla que estará compuesto por terrenos hexagonales.

2.2 Requisitos

Los requisitos de este proyecto se dividen en 4 tipos: funcionales, no funcionales, mecánicos y de interfaz.

Requisitos funcionales:

Desarrollar un juego de estrategia por turnos que satisfaga que:

- Existan distintos tutoriales que expliquen las mecánicas del juego de forma gradual.
- Exista un nivel completo donde el jugador deberá poder poner en práctica todo lo aprendido en los tutoriales para conseguir finalizar el objetivo que se le plantee.
- Durante la partida del nivel completo se pueda guardar el progreso para ser retomado en otro momento.
- Todos los productos jugables serán para un jugador que peleará contra la Inteligencia Artificial (IA). No tendrá modo multijugador u *online.*
- Tenga una IA funcional, está debe ser capaz de actuar y realizar distintas acciones para defenderse del jugador.
- El prototipo final debe encontrarse libre de errores (*bugs*).

Requisitos no funcionales:

- El prototipo será para PC.
- Deberá ser un prototipo que pueda ser ejecutado de forma fluida en ordenadores de gama baja - media con las siguientes características:
  - Disco duro: 1 GB - Ram: 2 GB
  - Windows: 7 o superior - Tarjeta gráfica: Nvidia GeForce GTX 650 o superior
  - Procesador: 2 GHz Dual Core (Intel Core 2 Duo 2.4 GHz) o superior
- El usuario debe tener una IU que le resulte cómoda.
- El prototipo debe ser fluido: debe tener una cantidad de FPS (*frames per second*) suficientes para que no dificulte la jugabilidad. Como mínimo 30 fps.
- Las mecánicas implementadas deberán estar libres de fallos y ser relevantes.
- Las unidades del juego deben estar balanceadas y ser relevantes.
- Tanto la interfaz de usuario (IU) como las unidades y terrenos del mapa deben ser visibles y comprensibles a simple vista.
- La IA debería funcionar de forma adecuada.
- Durante el proyecto se realizarán evaluaciones de las mecánicas e interfaces, por parte de personas externas al mismo usando para ello los tutoriales que se realicen.

Requisitos mecánicos

El género de videojuegos estrategia por turnos tiene una serie de mecánicas que lo diferencian de otros géneros. Estas mecánicas diferenciadoras diremos que son las mecánicas básicas de cualquier juego de este género. Por lo tanto el prototipo debe incluir dichas mecánicas (mecánicas comunes). El resto de mecánicas que se le añadan serán las mecánicas que no todos los videojuegos comparten (mecánicas diferenciadoras).

Mecánicas comunes:

Estas mecánicas deben ser las que sienten las bases del juego. Como ya se ha dicho no distan mucho de las mecánicas que hay en un tablero de ajedrez.

- Movimiento: Las unidades se podrán desplazar por el tablero para posicionarlas en la posición que el jugador requiera. Este movimiento será limitado por mecánicas avanzadas y se realizará siguiendo algún sistema de cálculo de caminos como Manhattan [11, 12].
- Combate: Las unidades podrán combatir con las unidades del bando contrario que se encuentren en su alcance. Este alcance será limitado por las mecánicas más avanzadas.
- Turno: Cada jugador (una IA es un jugador) podrá realizar distintas acciones con sus tropas y tras acabar se pasará el turno al siguiente jugador.

Mecánicas diferenciadoras:

Las siguientes mecánicas son las que deben hacer que nuestro prototipo tenga carácter y consiga una diferenciación con el resto de videojuegos del mercado.

- Piedra, papel o tijera: deben existir unidades de distintas clases y entre ellas debe existir una relación que haga que unas unidades sean más efectivas contra otras. Además estas unidades deben ser capaces de mostrarse útiles para distintas estrategias.
- Sanar a las unidades: las unidades no serán estáticas, podrán perder efectivos en las mismas (tras ser atacadas) por lo tanto el jugador debe ser capaz de rellenar las unidades con nuevas tropas para evitar su aniquilación.
- Niveles en las unidades: las unidades recibirán experiencia según acciones, con esta experiencia podrán subir de nivel y adquirir mejoras.
- Habilidades: las unidades podrán desbloquear la habilidad tras subir de nivel. Esta característica le proporciona a la unidad una forma alternativa de ataque. Cada tipo de unidad tendrá una habilidad característica.
- Formaciones: las unidades podrán desbloquear la formación tras subir de nivel. Esta característica le proporciona a la unidad una disposición alternativa de las tropas de la unidad. Las formaciones variarán la defensa, movilidad y ataque de la unidad. Cada unidad tendrá su formación característica.
- Ciclo solar: en el juego los días tendrán una duración de 3 turnos (mañana, tarde, noche). La luz variará dependiendo del momento, además hacer que las tropas que se muevan de noche tendrá consecuencias negativas. Otra característica será que cuando una unidad pasa un dia completo (3 turnos) sin descansar recibirá una penalización que se irá incrementando si no descansa en los siguientes turnos.
- Esperar: las unidades podrán eliminar el cansancio usando esta acción pero se bloquearan el resto de posibles acciones de la unidad.
- Mapa dinámico: el tablero podrá modificarse durante la partida apareciendo o desapareciendo componentes.
- Aparición de tropas: deben existir casillas capaces de poner tropas en el tablero cuando suceda un evento determinado (por ejemplo una unidad del jugador alcanza una casilla objetivo). Tanto el jugador como la IA podrán recibir más unidades durante la partida a través de estas casillas. Estas casillas se conocen como *spawns.*
- Niebla de guerra: típica mecánica de este género de videojuegos que hace que el jugador no vea de un inicio todo el tablero y deba visualizarlo usando las unidades para ello.

Reglas mecánicas:

Los elementos que vienen a continuación no son mecánicas ya que no cambian el *game state* pero son reglas que afectan a las mecánicas ya explicadas.

- Unidades de distintas clases: las unidades con las que se cuenten deberán ser de distintas características para aumentar las posibilidades estratégicas del jugador a la hora de enfrentarse al reto que se le ofrezca. Esta regla afecta a las mecánicas de “combate”, “piedra, papel, tijera”, “niebla de guerra” y “movimiento”.
- Valor de movimiento: las unidades tendrán un valor de movimiento, o lo que es lo mismo solo podrán moverse dependiendo de ese valor. Dependiendo del tipo de unidad y el nivel este valor variará. Esto interactúa con la mecánica de “movimiento”.
- Visión de las unidades: dependiendo del tipo de unidad variará el valor de visión. Cuanto mayor sea el campo de visión más niebla de guerra despejaran del tablero. Esto interactúa con la mecánica de “niebla de guerra”.
- Casillas de distintas clases: Las casillas que componen el tablero deberán ser de distintas clases. Cada clase tendrá unas cualidades diferenciadoras, estas cualidades tendrán importancia en el *gameplay* del jugador ya que modificarán los valores de las unidades.
- Bonus de casilla: las casillas modifican la defensa y el ataque de la unidad que esté sobre ella. El bonus recibido podrá ser tanto negativo como positivo dependiendo del tipo de casilla. Esta regla afecta a la mecánica de “combate”.
- Coste de movimiento de casilla: el desplazamiento por las casillas tendrá un valor determinado dependiendo de su tipo. Esto interactúa con la mecánica de “movimiento”.

Requisitos de interfaz:

Como es lógico no todo el sistema se sustentará en base a las mecánicas, deben existir elementos de interfaz que permitan al usuario interactuar con el juego de forma adecuada.

- Cámara libre: la cámara del usuario debe ser lo suficientemente móvil y fluida para que el usuario pueda desplazar la vista por el tablero sin que le resulte incómodo.
- La cámara debe moverse de forma fluida y adecuada a lo indicado por el usuario.
- Durante la partida el usuario pueda consultar datos de los terrenos y unidades cuando lo desee, visualizando estos datos a través de la IU .
- El usuario debe disponer de una segunda cámara para visualizar el terreno desde una perspectiva aérea.
- Diferenciación visual de las unidades: las unidades deben poder diferenciarse entre las distintas clases con solo mirarlas en el tablero, por lo tanto deberán tener una representación física distinta.
- IU de acción: cuando se seleccione una unidad este debe tener un elemento en la IU que permita decidir qué acción quiere seleccionar el jugador.
- Minimapa: existirá una cámara y un elemento en la IU que muestre al usuario una vista aérea del tablero en todo momento para mejorar su perspectiva del tablero.
- Mensajes al jugador: el juego deberá poder mostrar mensajes informativos al usuario.
- Datos de unidades: el usuario debe ser capaz de visualizar los datos de sus unidades durante la partida. Para ello existirá un elemento en el IU que muestre los datos de la unidad seleccionada.

- Visualización del entorno: las características del terreno deben ser visibles a simple vista con un conjunto de colores que identifique la zona.
- Datos del terreno: el usuario debe ser capaz de visualizar los datos y bonuses que ofrece un terreno (casilla). Para ello existirá un elemento en el IU que muestre los datos de la casilla seleccionada.
- Menú principal: el jugador al encender el juego deberá tener un menú para elegir alguna opción. Estas opciones pueden ser la elección entre distintos tutoriales o niveles y la opción de salir del juego.
- Menú de pausa: el jugador debe disponer durante la partida de un menú con el que pueda decidir si salir o guardar la partida y pare la acción del juego durante esta pausa.

2.3 Metodología

Dado que existe un desconocimiento sobre cómo afrontar el proyecto, este se dividirá en bloques de implementación que puedan ser modificados a lo largo del proyecto. Para ello se usará una metodología AGILE o por lo menos se usarán características de las mismas. En concreto se emulará a los *sprints* de la metodología Scrum [13] con dos pretextos:

Implementación
Para la realización de la implementación se propondrá una planificación inicial donde se dirá en qué orden se implementaran las mecánicas descritas en los requisitos. Aunque dependiendo de lo dicho por usuarios que testeen el juego o por problemas técnicos, se podrá modificar este orden o se eliminarán mecánicas de los sprints posteriores. Las mecánicas que se implementarán son las descritas anteriormente.

Entregables de sprint
Los sprints tendran una duracion 3 semanas. Al final de cada ciclo se deberá tener un prototipo que contendrá una explicación de las mecánicas desarrolladas en ese ciclo. Los prototipos serán mostrados a unos testeadores en forma de escenarios donde se les explicara y se les dejara probar las mecánicas implementadas en el sprint.

Estos prototipos serán de alta fidelidad (*hi-fi*) y presentarán un modelo completo del producto final con funcionalidad real, aunque no con toda la funcionalidad.

Los testeadores serán voluntarios que probaran los distintos productos provenientes de los *sprints.* Al ser voluntarios puede que a lo largo del proyecto cesen de realizar los testeos. Al comenzar el proyecto hay por lo menos 5 voluntarios. Los voluntarios son todos jugadores avanzados, lo que posiblemente haga que proporcionen opiniones de calidad. Además entre estos testeadores hay jugadores acostumbrados a juegos del género del prototipo y otros son novatos en el género. Con esto se quiere intentar que exista una variedad de opiniones.

Estos testeos se realizan por cuatro razones:

1 Este proyecto se centra en las mecánicas del juego, por lo que estas deben ser robustas y no realizar comportamientos extraños que interfieran con la siguiente capa de mecánicas que se añadirán en el ciclo posterior. Además, que personas ajenas al desarrollo prueben el prototipo ayudará a encontrar errores.

2 “Mínimo producto viable” [14]. Al leer un poco sobre desarrollo de videojuegos se puede observar que este término aparece de manera frecuente. En esencia quiere decir que hay que tener siempre un prototipo que pueda mostrar qué es y hacia dónde va el proyecto sin dejar muchas cosas a la imaginación.

3 Será una buena práctica para recoger información de testeos y de cómo los futuros jugadores ven las mecánicas. Con ello se podrán balancear las mecánicas para que se integren de forma correcta unas con otras.

4 Obtener un buen ritmo en el desarrollo y no tener retrasos con la implementación. El tener que sacar productos viables que sean juzgados cada 3 semanas ayudará a esta tarea.

Al final de cada *sprint* se enviará el prototipo a los 5 testeadores, se le adjuntará una encuesta (test) que los testeadores rellenaran. Las preguntas de estas encuestas pedirán que respondan principalmente a 3 temas:

- Si el testeador encuentra errores.
- Si alguna mecánica no se entiende bien.
- Si la interfaz es útil y se entiende.

Tras cada pregunta de tipo test sobre la conveniencia de una mecánica existirá una pregunta donde el usuario podrá escribir sus impresiones sobre lo antes preguntado si lo ve conveniente. Esto se hará porque las preguntas tipo test no tienden a dar información de mucha profundidad pero sí ayudan a ver donde hay problemas.

Además los testeadores tendrán siempre una pregunta donde podrán contestar lo que ellos vean necesario. Este “cajón de sastre” puede volverse muy útil si los testeadores están muy motivados a colaborar.

Además de estos testeos, se realizará un *thinking aloud [15]* donde se intentarán sacar a la luz los temas que se traten en las preguntas “cajón de sastre”, esto se hará con el objetivo de poner en común estos temas y que se debatan entre los testeadores.

Thinking aloud

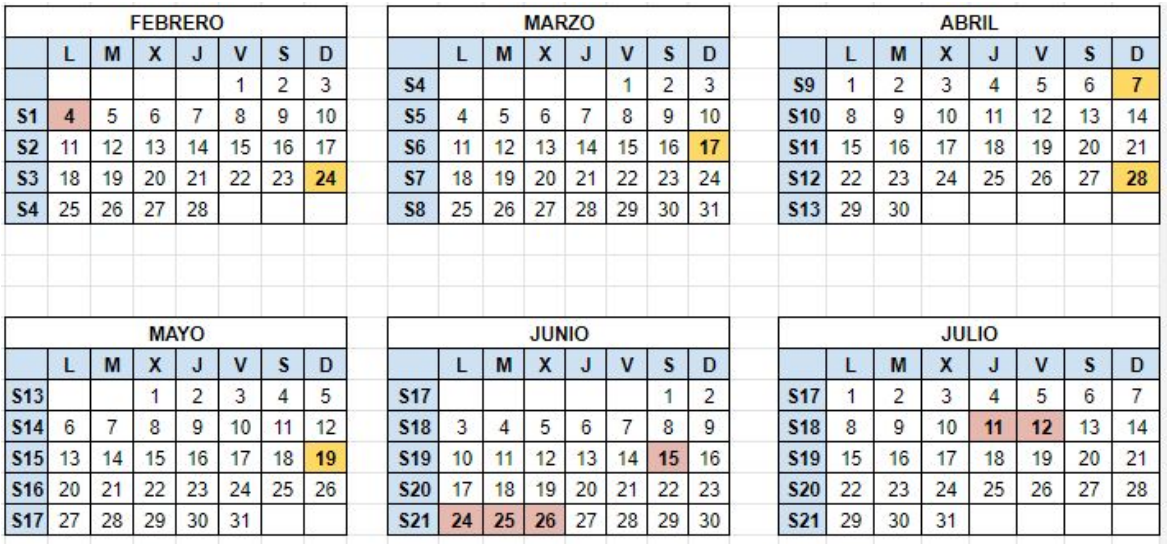
El *thinking aloud* es un tipo de test que permite recabar gran cantidad de información de los testeadores de forma más eficiente que simples tests. Para su realización se selecciona a una cantidad de testeadores y se les reúne en una sala, un examinador recoge por escrito

las distintas ideas y pensamientos explicados por los testeadores mientras usan la aplicación a probar.

La idea de este thinking aloud será sacar mejoras en la interfaz y mejoras para la mejor comprensión de las mecánicas. Por eso se realizará en un sprint avanzado (final del 3 sprint o 4 sprint), el en cual se tenga casi todas las mecánicas implementadas y los testeadores ya estén familiarizados con el juego.

2.4 Calendario e hitos

En la imagen 8 se muestra el calendario que se seguirá durante el TFG. En él se pueden observar la disposición cronológica de los distintos hitos



FEBRERO							
	L	M	X	J	V	S	D
					1	2	3
S1	4	5	6	7	8	9	10
S2	11	12	13	14	15	16	17
S3	18	19	20	21	22	23	24
S4	25	26	27	28			

MARZO							
	L	M	X	J	V	S	D
S4					1	2	3
S5	4	5	6	7	8	9	10
S6	11	12	13	14	15	16	17
S7	18	19	20	21	22	23	24
S8	25	26	27	28	29	30	31

ABRIL							
	L	M	X	J	V	S	D
S9	1	2	3	4	5	6	7
S10	8	9	10	11	12	13	14
S11	15	16	17	18	19	20	21
S12	22	23	24	25	26	27	28
S13	29	30					

MAYO							
	L	M	X	J	V	S	D
S13			1	2	3	4	5
S14	6	7	8	9	10	11	12
S15	13	14	15	16	17	18	19
S16	20	21	22	23	24	25	26
S17	27	28	29	30	31		

JUNIO							
	L	M	X	J	V	S	D
S17						1	2
S18	3	4	5	6	7	8	9
S19	10	11	12	13	14	15	16
S20	17	18	19	20	21	22	23
S21	24	25	26	27	28	29	30

JULIO							
	L	M	X	J	V	S	D
S17	1	2	3	4	5	6	7
S18	8	9	10	11	12	13	14
S19	15	16	17	18	19	20	21
S20	22	23	24	25	26	27	28
S21	29	30	31				

Imagen 8: *calendario de los meses que abarca el TFG y los hitos más significativos*

Las fechas marcadas en amarillo son los hitos que muestran que día se acabarán los distintos *sprints*, como se ha comentado anteriormente cada *sprint* tiene una duración de 3 semanas. Como ya se indicó, el objetivo es tener un prototipo funcional (de las mecánicas introducidas) al acabar cada uno de ellos.
Las fechas marcadas en rojo indican el comienzo, entrega y defensa del TFG ver tabla 1.

Fecha	Representa
4 de febrero	Comienzo del TFG
24 de febrero	Fin del primer sprint
17 de marzo	Fin del segundo sprint
7 de abril	Fin del tercer sprint
28 de abril	Fin del cuarto sprint
19 de mayo	Fin del quinto sprint
15 de junio	Fin del TFG
24 al 26 de julio	Depósito TFG
11 o 12 de julio	Defensa del TFG

Tabla 1: *correspondencia entre fechas e hitos*

2.5 Estructura de descomposición de trabajo

En la imagen 9 se muestra el gráfico de la EDT

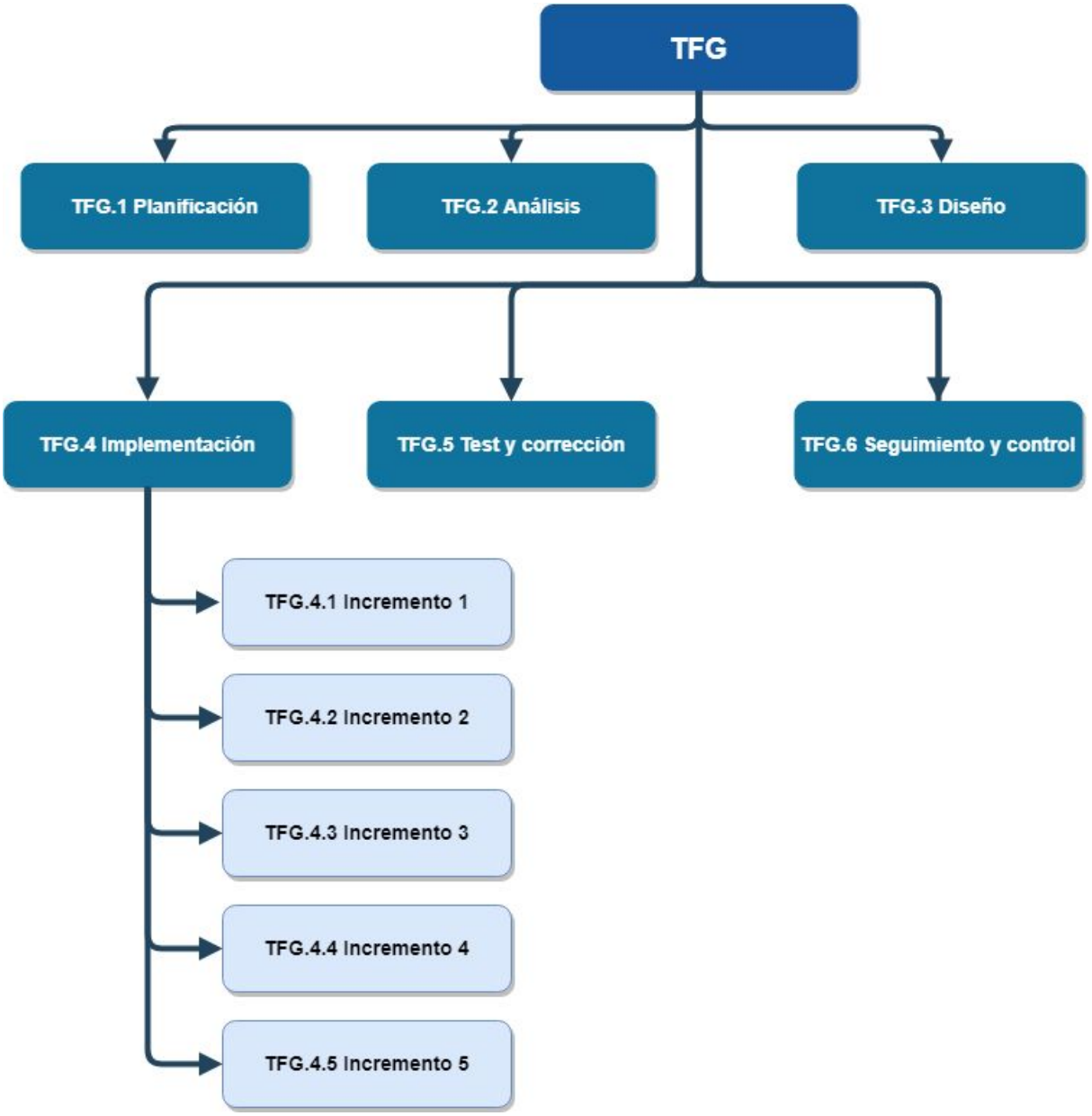


Imagen 9: *árbol EDT*

2.6 Diccionario de la EDT

En las tablas 2 y 3 se muestran las distintas descomposiciones de la EDT.

Tarea		Entregable
TFG.1	**Planificación**	Apartado 2 memoria y anexo 1 ”Re-planificación”.
TFG.2	**Análisis**	Apartado 3 memoria y anexo 2” Especificación de mecánicas y casos de uso”.
TFG.3	**Diseño**	Apartado 4 memoria y anexo 3” Diseño”.
TFG.4	**Implementación**	Apartado 5 memoria y anexo 4 “*Sprints*”.
TFG.4.1	**Incremento 1**	Apartado 6.1 memoria y Tutorial 1.
TFG.4.2	**Incremento 2**	Apartado 6.2 memoria y Tutorial 2.
TFG.4.3	**Incremento 3**	Apartado 6.3 memoria y Tutorial 3.
TFG.4.4	**Incremento 4**	Apartado 6.4 memoria y Nivel 1.
TFG.4.5	**Incremento 5**	Apartado 6.5 memoria y Nivel 1 versión 2.
TFG.5	**Test y corrección**	Apartado 6 de la memoria y anexo 5 “*Thinking aloud*” .
TFG.6	**Seguimiento y control**	Apartado 6 de la memoria.

Tabla 2: *biblioteca de la EDT*

Tarea		Descripción
TFG.1	**Planificación**	Apartado donde se indicarán los distintos hitos del TFG. También se indicarán qué tareas se llevarán a cabo y la duración de las mismas.
TFG.2	**Análisis**	Apartado donde se investigan las herramientas que se usarán en el TFG y se explicara las mecánicas que se implementaran. Además se modelaron los casos de uso.
TFG.3	**Diseño**	Apartado donde se diseñan los distintos componentes del juego y sus características.
TFG.4	**Implementación**	Apartado donde se detalla el proceso de implementación de los distintos sprint.
TFG.4.1	**Incremento 1**	Implementación del primer sprint.
TFG.4.2	**Incremento 2**	Implementación del segundo sprint.
TFG.4.3	**Incremento 3**	Implementación del tercer sprint.
TFG.4.4	**Incremento 4**	Implementación del cuarto sprint.
TFG.4.5	**Incremento 5**	Implementación del quinto sprint.
TFG.5	**Test y corrección**	Apartado donde se realizarán los test y las correcciones de los errores encontrados por los usuarios en los prototipos.
TFG.6	**Seguimiento y control**	Apartado donde se indicarán los impedimentos que han surgido a lo largo del proyecto y la duración de los procesos.

Tabla 3: *explicación de los distintos apartados de la EDT*

2.7 Diagrama Gantt

En la imagen 10 se muestra el diagrama de Gantt, donde se describe cómo se repartirá la carga de trabajo a lo largo del proyecto.

			SEMANAS															
EDT	Tarea	Horas estimadas	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17
TFG.1	Planificación	10																	
TFG.2	Análisis	10																	
TFG.3	Diseño	10																	
TFG.4	Implementación	200																	
TFG.4.1	Incremento 1	50																	
TFG.4.2	Incremento 2	40																	
TFG.4.3	Incremento 3	50																	
TFG.4.4	Incremento 4	30																	
TFG.4.5	Incremento 5	30																	
TFG.5	Realización y análisis de tests	25																	
TFG.6	Seguimiento y documentación	35																	
		290																	

***Imagen 10**: diagrama de Gantt del proyecto completo*

Como puede verse en el diagrama de Gantt en las primeras 4 semanas la carga de trabajo es de gran intensidad. Esto se debe a que el primer mes se dispone de mucho tiempo para realizar el TFG, además esto ayudará a no tener problemas mas adelante evitando posibles retrasos y prisas los últimos días.

Cada *sprint* tendrá una serie de tareas asignadas, estas tareas serán mayoritariamente mecánicas. Como se ha dicho, el resultado final de cada sprint será un prototipo que será testeado. Además se generará la documentación explicando cómo se resolvieron las tareas realizadas a lo largo del sprint. Por lo tanto cada sprint tendrá un subproducto compuesto por:

- Prototipo.
- Test del prototipo.
- Documentación del sprint.

Cada sprint tendrá un cometido. En el caso del primero se comenzará a implementar durante el desarrollo de la planificación, diseño y análisis. Esto se debe a que se usará este *sprint* como aprendizaje de las herramientas que se usarán para el desarrollo del proyecto. Esto permitirá acotar mejor la planificación y el diseño. Durante este aprendizaje se espera realizar las mecánicas referentes al movimiento. En la tabla 4 se pueden ver las distintas tareas que se implementaran en el primer sprint.

		SEMANAS		
Primer Sprint	**Horas estimadas**	**S1**	**S2**	**S3**
Mapa	4			
Tipos de suelo	4			
Movimiento (posibles movimientos)	8			
Movimiento (Camino más corto)	8			
Unidades aliadas	4			
IU minimapa mensajes y menús básicos	8			
Pantalla de Inicio	6			
Primer tutorial	8			
Total:	50			

Tabla 4: *diagrama de Gantt de las tareas que se realizarán en el primer sprint.*

En este Gantt solo se reflejan las tareas relacionadas con la implementación. El tiempo dedicado a la documentación del *sprint* y a la creación de los test está englobado en las tareas de "seguimiento y documentación" y "realización y análisis de tests" del gantt del proyecto.

La definición del resto de *sprints* se puede ver en el anexo "sprints". En él se detalla el objetivo del sprint, las tareas asignadas a cada uno y la resolución de las mismas.

2.8 Plan de riesgos

Las tablas 5, 6, 7 y 8 muestran los distintos tipos de riesgos existentes en el TFG y las distintas medidas que se realizarán para evitarlos.

Riesgos con la Tecnología			
Riesgo	**Estrategia de prevención**	**Estrategia de minimización**	**Plan de contingencia**
Inexperiencia con la tecnología utilizada	Leer documentación y datos previamente a la realización de la implementación.	La misma que en prevención.	Consultar foros, libros y tutoriales sobre los temas que desconozca.
Poca eficiencia con la tecnología a usar los primeros días	Informarme de atajos, técnicas y consejos sobre cómo usar las herramientas.	La misma que en prevención.	Disponer de un margen de horas mayor para las primeras etapas de implementación.

Tabla 5: *riesgos con la Tecnología*

Riesgos con el alcance del producto			
Riesgo	**Estrategia de prevención**	**Estrategia de minimización**	**Plan de contingencia**
Alcance poco claro	Definir un alcance mínimo suficiente para tener un prototipo final con una cantidad de mecánicas suficiente.	Indicar mecánicas suficientemente grandes en el alcance inicial para que el producto sea jugable.	Realizar un reajuste de la planificación si se ve que el alcance inicial no era realista.
Alcance demasiado optimista	Evitar poner un alcance excesivo en el plan inicial, pero que sea suficiente.	Las mecánicas se implementaran en orden de importancia para que aunque no se complete todo el proyecto, exista un prototipo final jugable (El del anterior sprint).	Los prototipos serán jugables. Por lo tanto si es imposible avanzar a partir de un sprint determinado, se tendrá un prototipo presentable (El del anterior sprint).
Cambios en el alcance	Se prevé que puedan existir cambios en las mecánicas a implementar.	Las mecánicas se implementaran en orden de importancia para que aunque no se llegue al plan inicial exista un prototipo final jugable (El del anterior sprint).	Cada prototipo será jugable y tendrá unas mecánicas muy bien definidas por lo que si se quieren cambiar solo afectará al último prototipo y los anteriores seguirán siendo funcionales.

Tabla 6: *riesgos con el alcance del producto*

Riesgos con el desarrollo del proyecto			
Riesgo	**Estrategia de prevención**	**Estrategia de minimización**	**Plan de contingencia**
Pérdida de documentación	La documentación se realizará íntegramente en drive, ofrece la recuperación de los documentos borrados.	La misma que en prevención.	Periódicamente se descargan los archivos desde drive y se almacenarán en un disco duro. En caso de necesidad restaurarlos con esa copia.
Pérdida de código proyecto	El ordenador donde se programa es fijo (no se moverá). Esto disminuye la probabilidad de que el dispositivo se pierda junto a la información. Además se realizarán 2 copias de seguridad diarias.	Evitar tocar la carpeta del proyecto o moverla de su ubicación durante el desarrollo del trabajo.	En el caso del borrado, el directorio que contiene el proyecto es copiado cada dia a 2 discos duros independientes. por lo que reside en 2 sitios distintos desde los que se puede recuperar.

Tabla 7: *riesgos con el desarrollo del proyecto*

Riesgo	Estrategia de prevención	Estrategia de minimización	Plan de contingencia
Ausencia del tutor	No se puede prevenir.	Mantener comunicación con el tutor periódicamente.	Mientras no se tenga comunicación con el tutor seguir según el rumbo fijado en la planificación hasta nueva orden.
Insatisfacción con el avance del proyecto	Definición de estrategias que obliguen a realizar un trabajo mínimo en un tiempo. Prototipos y Scrum.	Centrarse en el prototipo en desarrollo para poder sacarlo el día indicado.	No se puede crear un plan de contingencia.
Insatisfacción con la calidad del proyecto	No tener grandes expectativas iniciales.	Pensar solo en los ciclos cercanos y no preocuparse del futuro del producto, solo de la calidad del prototipo que se esté realizando en ese momento.	Usar los prototipos para que otros individuos puedan ver y buscar fallos. Esto ayudará a la mejora de la calidad.

Tabla 8 : *riesgos con el tutor*

3. Análisis

En esta sección se abordará la especificación de las mecánicas a desarrollar, los videojuegos que han servido como inspiración para ellas y el estudio de la interacción del usuario con el sistema a través de diagramas de casos de uso.

3.1 Videojuegos usados como referencia

Tras tener las herramientas que usaremos durante el proyecto, pasamos a analizar qué juegos podemos usar como referencia para obtener ideas de qué mecánicas implementar, cuales no y su interacción entre ellas. Se ha usado como referencia distintos juegos ya existentes. El producto de este TFG es una agrupación de distintas mecánicas existentes en juegos del género y una serie aportaciones, que hasta lo que conozco, no habían aparecido en otros videojuegos.

A continuación se mostrará un ejemplo de mecánica, de interfaz y de idea no observada en otros juegos del género.

Mecánica. Habilidades en las unidades:
En algunos juegos de estrategia por turnos se ofrece al jugador que sus unidades puedan utilizar una habilidad para poder cambiar el curso de un enfrentamiento. Estas habilidades pueden ser desde un ataque nuevo (como las habilidades de las unidades de este TFG) a mejoras de movilidad o cambios de estado del *game state* (por ejemplo que llueva en el tablero y esto tenga consecuencias sobre todas las unidades). Un ejemplo de juego que implementa habilidades es "*Mario + Rabbids Kingdom Battle*", ver imagen 11.

Imagen 11: *juego Mario + Rabbids Kingdom Battle. Puede verse en la parte inferior de la pantalla la selección de movimientos y habilidades que dispone la unidad.*

Interfaz. Casillas con forma hexagonal:
Normalmente las casillas que forman el tablero por el que las unidades se desplazan tienen forma de cuadrado. En el caso de este TFG no se usan cuadrados sino hexágonos. La idea de esto viene dada por el juego *"Panzer tactics HD"*, ver imagen 12.

Imagen 12: *uso de casillas hexagonales en el juego "Panzer tactics HD"*

Idea no observada. Cansancio y ciclo solar:
En los juegos con los que he tenido contacto no he encontrado ningun juego que penalice al jugador por usar demasiadas veces seguidas a una unidad (y no dejarla descansar), en algunos juegos se deben gestionar recursos como la munición o el combustible usado para desplazarse, pero no se tiene en cuenta el desgaste por cansancio que sufren las unidades. Para reforzar la idea del cansancio se implementa la mecánica del ciclo solar donde la luz del mapa va cambiando según el turno dando sensación de que pasa el tiempo.

3.2 Especificación de las mecánicas

Tras tener claras las mecánicas que se quieren implementar, hay que organizarlas y explicar su comportamiento. En este apartado se explican las distintas mecánicas indicadas en los requisitos. En este documento se pondrá un ejemplo de una mecánica común, una diferenciadora y una regla, ver tablas 9, 10 y 11. La especificación del resto de mecánicas se proporciona en el anexo "Especificación de mecánicas y casos de uso".

Mecánica común:

Mecánica	**Combate.**
Objetivo	Las unidades podrán combatir entre ellas.
Precondición:	La unidad ha tenido que ser seleccionada. La unidad solo podrá atacar si no ha realizado una acción de combate y no ha esperado o sanado en el turno. El usuario ha debido seleccionar la unidad enemiga a la que quiere atacar, esta selección estará limitada por el alcance del ataque de la unidad atacante.
Postcondición	Tras el enfrentamiento las dos unidades calcularán su salud que será su salud inicial - (daño recibido - daño absorbido por las armaduras). Las dos unidades recibirán daño del enemigo durante el combate, pero la defensora realizará un porcentaje menos daño para premiar al jugador agresivo.

Tabla 9: *especificación de una mecánica común*

Mecánica diferenciadora:

Mecánica	**Ciclo solar.**
Objetivo	Alterar las características de las unidades dependiendo del turno en el que nos encontremos. Obligar al usuario a pensar cómo se encontraran sus tropas en los turnos siguientes.
Precondición:	Finaliza un turno y se activa el ciclo solar.
Postcondición	La posición del foco de luz (sol) del tablero habrá cambiado a una de sus tres posibles posiciones, estas posiciones representan el sol al amanecer, atardecer y anochecer. Si el nuevo estado es anochecer, todas las tropas reciben un bonus negativo del 20% del ataque durante ese turno. Además todas las unidades actualizan su valor de cansancio aumentando por cada turno que no descansen. Cuanto mayor sea este valor menos efectivas serán las unidades.

Tabla 10: especificación de una mecánica diferenciadora

Regla:

Regla	**Casillas de distintas clases.**
Objetivo	Que el jugador tenga que realizar un esfuerzo mental mayor a la hora de afrontar la distribución de las unidades en el tablero, esto se debe a que las distintas casillas cambiarán el rendimiento de las tropas que se encuentren sobre ellas dependiendo del tipo de casilla.
Implicación	El movimiento y el combate de las unidades de los jugadores se verá modificado por los valores de las casillas que interactúen con la unidad.

Tabla 11: especificación de una regla

3.3 Modelo de caso de uso

A continuación se muestran los diagramas de casos de uso que expresan las acciones que puede realizar el usuario en el menú principal del juego (imagen 13) y durante la partida (imagen 14).

El usuario en el menú principal

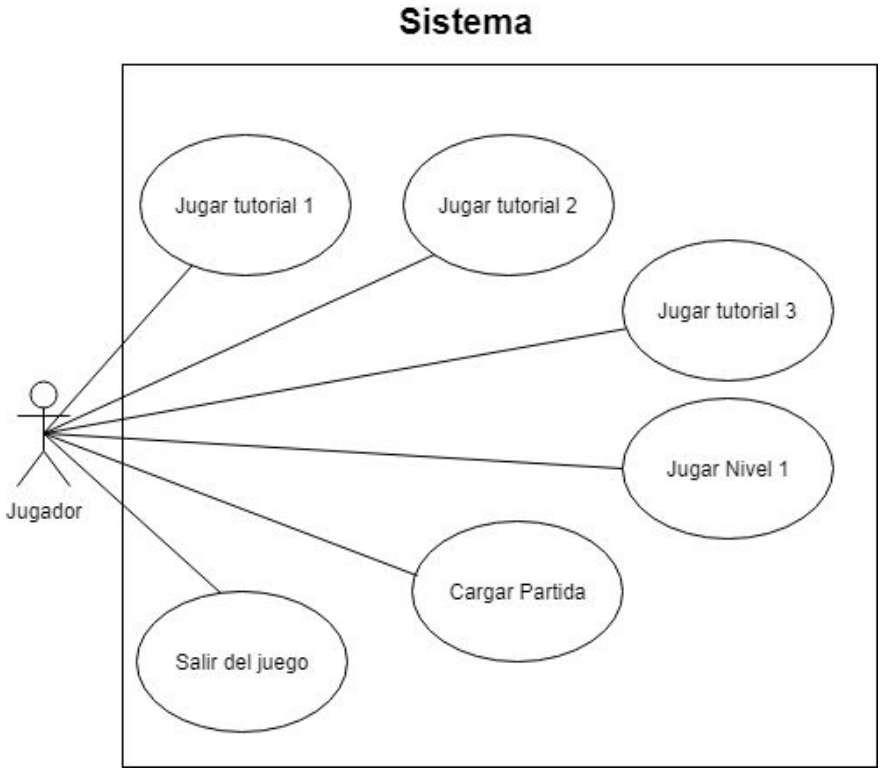


Imagen 13: *casos de uso en el menú principal*

El usuario durante la partida

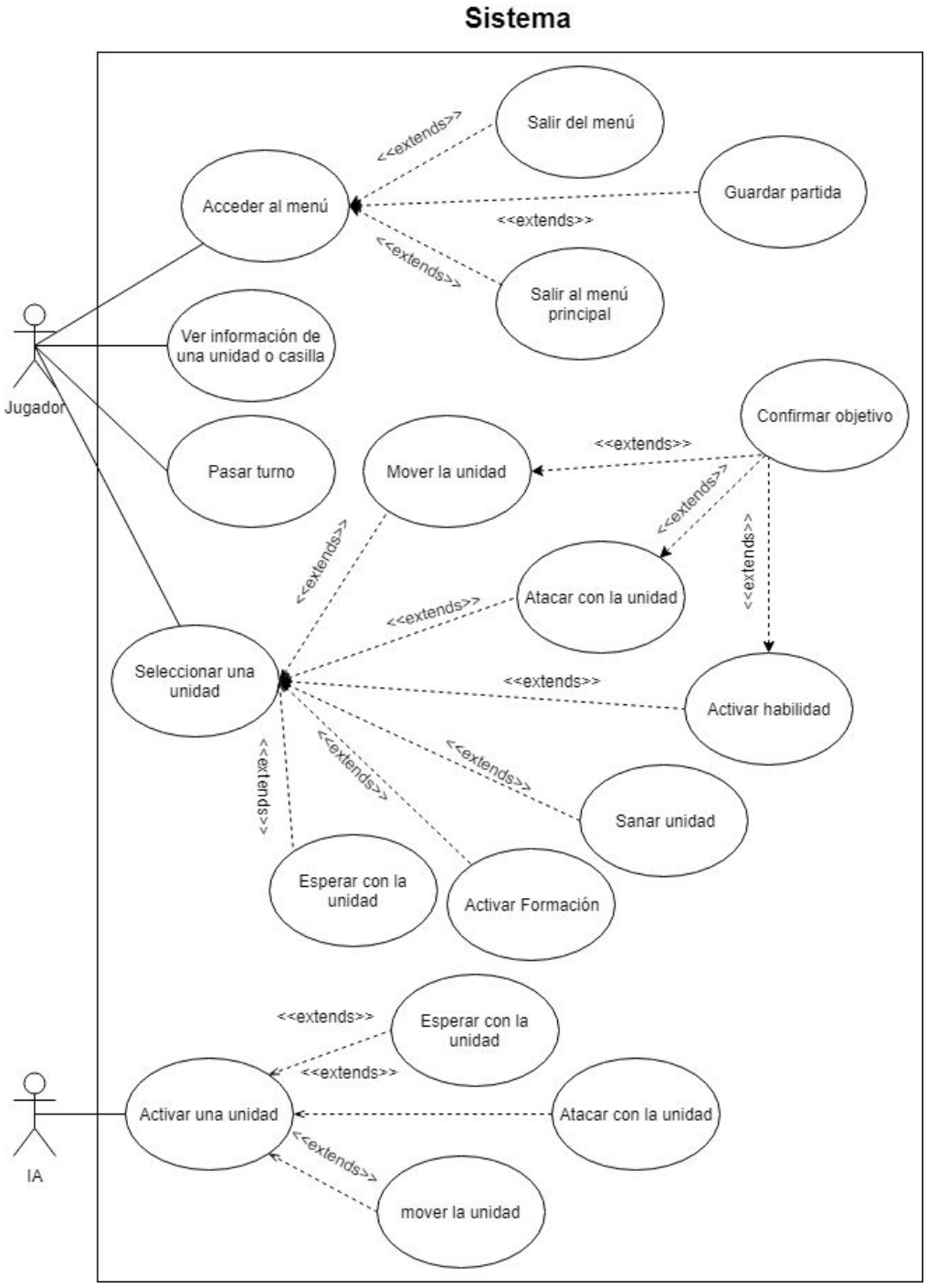


Imagen 14: *casos de uso durante la partida*

3.4 Especificación de caso de uso

Las tablas mostradas a continuación son un ejemplo de las especificaciones de todos los casos de uso. El resto de casos de uso se encuentran en el anexo "Especificaciones de mecánicas y casos de uso".

Nombre:	**Seleccionar una unidad.**
Actores:	Jugador.
Precondición:	Debe ser el turno del jugador.
Descripción:	El jugador activa una unidad y se muestran los paneles de las acciones que tiene disponibles la unidad.

Tabla 12: *especificación caso de uso "seleccionar una unidad"*

Nombre:	**Mover una unidad.**
Actores:	Jugador.
Precondición:	La unidad debe ser seleccionada y debe poderse mover.
Descripción:	Se mostrará a qué casillas se puede desplazar la unidad.

Tabla 13: *especificación caso de uso "mover una unidad"*

Nombre:	**Pasar turno.**
Actores:	Jugador.
Precondición:	Debe ser el turno del jugador.
Descripción:	El jugador cede el turno y la IA comienza su turno. Todas las unidades del jugador pasan a estar usadas.

Tabla 14: *especificación caso de uso "pasar turno"*

4. Diseño

En este apartado se explica el diseño inicial de las distintas partes del proyecto, interfaces, componentes (unidades, casillas) y sus características. Para evitar problemas de espacio en este documento solo se pondrán ejemplos de los distintos apartados, en el anexo "Diseño" se encontrará la totalidad del diseño.

Todos estos diseños son iniciales. Por lo que podrán sufrir cambios a lo largo del proyecto. Estos cambios se verán reflejados en el apartado "incrementos" o en el anexo "*sprints"*.

4.1 Interfaces

Como se ha ido explicando, este TFG tiene como objetivo realizar una cantidad importante de mecánicas. Esto no implica que haya que descuidar la interfaz, ya que tiene que ser cómoda y entendible para que los futuros jugadores puedan interactuar cómodamente con ella.

Para ello se seguirán varias máximas que se respetan todo lo posible:

- Todo lo que no deba centrar la atención del jugador estará puesto lejos del centro de la pantalla.
- El usuario tendrá que disponer siempre de información. Por lo tanto tiene que existir la posibilidad de poder leer siempre una explicación de que está señalando con el ratón.
- La interfaz debe ser minimalista para evitar sobrecargar al jugador con información.
- Los colores y formas deben ser fáciles de ver y comprender aunque la calidad del apartado gráfico es baja (no nos centramos en ello en este proyecto).

Aun con todo esto para mejorar la interfaz y la comprensión de las mecánicas se realizarán distintos tipos de test que ayudarán a lo largo del proyecto a pulir el diseño.

Las imágenes 15 y 16 son diseños iniciales de la interfaz y dan una idea aproximada de qué forma tendrá que tener la interfaz final del prototipo.

Menú

Imagen 15: *diseño del menú principal del juego*

Al iniciar el juego lo que debe aparecer es este menu que contenga 6 botones:

- Los 3 primeros son para acceder a los tutoriales 1, 2 y 3.
- El cuarto para acceder al primer nivel de la campaña.
- El quinto para cargar la partida que ha sido guardada con anterioridad.
- El sexto para salir del juego.

Durante la partida

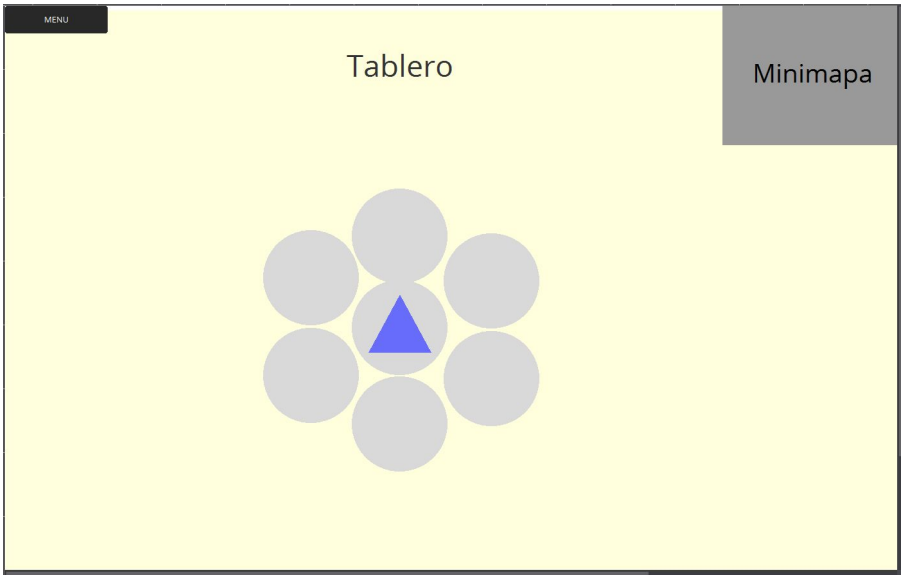


Imagen 16: *diseño de la vista del usuario durante la partida*

Durante la partida lo que ocupará toda la pantalla será una vista isométrica del tablero que estará compuesta por casillas hexagonales (representadas aquí con círculos) y las unidades serán objetos sobre las mismas (representada aquí con un triángulo).

En la parte superior derecha se encontrará el minimapa que mostrará una vista aérea del mapa sobre la posición del jugador.

En la parte superior izquierda se encontrará un botón que desplegará el menú de pausa.

El resto de interfaces diseñadas se encuentran en el documento de diseño.

4.2 Unidades, terrenos y datos de las mismas

El usuario además de poder interactuar con la interfaz podrá interactuar con las unidades y estas a su vez interactúan con las casillas que componen el mapa. Por lo tanto los elementos principales del juego serán las casillas y las unidades.

Casillas

Las casillas son una representación de un territorio con unas características que afectarán a las unidades que están en ellas o las crucen. Las características que una casilla puede tener son:

- **Coste de movimiento:** coste que tiene cruzar este territorio, si la tropa no tiene una capacidad de movimiento igual o superior, no podrá cruzar el territorio.
- **Bonus defensivo:** Añade un porcentaje de defensa a la unidad que se situé sobre ella.
- **Bonus ofensivo:** Añade un porcentaje de ataque a la unidad que se situé sobre ella.
- **Bonus de visión:** Añade puntos de visión a la unidad que se situé sobre ella.
- **Monedas**: El cuartel es la única casilla que tiene esta cualidad. Las monedas sirven para curar a las unidades. Cada turno el jugador recibirá 1 moneda por cada cuartel que controle.

En la tabla 15 se indican las características de los distintas casillas. (estas características podrán variar en función de los futuros balanceos).

Tipo de casilla	Coste de movimiento	Bonus defensivo	Bonus ofensivo	Bonus de visión
Cuartel	1	0	0	0
Camino	0.75	-0,1	0	0
Pradera	1	0	0	0
Arboleda	1.25	0,15	-0,1	0
Ciudad	1	0,1	0,1	0
Fortín	3	0,5	0,3	1
Rio	8	0	0	0
Terreno elevado	2	0,2	0,2	1

Tabla 15: *valores de las distintas casillas*

Unidades

Las unidades son la representación de las fichas del jugador y de la IA, existirán 4 tipos: Escuderos, Lanceros, Arqueros y Caballeros. Los distintos tipos se agruparán en unidades que podrán ser de 1 a 25 soldados, las unidades empezarán con 25 pero tras combatir disminuirán y se podrá recuperar el valor máximo sanando a la unidad. Si el número llega a 0 la unidad es eliminada. Los caballeros tendrán un máximo de 20 unidades. Los datos puestos a continuación son los valores que se darán a las unidades durante el segundo sprint. Se espera que estos valores puedan variar tras el testeo de ese sprint y se balanceen las unidades si alguna es demasiado poderosa, ver tablas 16 y 17.

Tipo de unidad	Ataque	Defensa	Salud
Escudero	0.75	1	1
Lancero	1	0.5	1
Arquero	0.5	0.35	1
Caballero	1.5	0.25	1

Tabla 16: *valores de un soldado en cada de cada tipo de unidad*

Tipo de sección	Movimiento	Visibilidad	Rango de ataque
Escudero	2 uni.movimiento	2 casillas	1 casillas
Lanceros	2 uni.movimiento	2 casillas	1 casillas
Arqueros	2.5 uni.movimiento	3 casillas	2 casillas
Caballeros	3 uni.movimiento	4 casillas	1 casillas

Tabla 17: *valores de las distintas Unidades*

Habilidades y formaciones

Cada tipo de unidad tendrá una formación y una habilidad que podrán usar, estas variarán dependiendo del la unidad. La habilidad y la formación se conseguirán cuando la unidad aumente de nivel.

- Escuderos:

Habilidad:

Nombre: Pilum.
Daño efectuado: 1 punto de daño por tropa en la unidad.
Distancia de ataque: 2 casillas.
Disminuye la defensa enemiga en un 33% de forma permanente.

Formación:

Nombre: Tortuga.
Plus defensivo: 50%.
Penalización:
Reducción de movimiento en 1.
Reducción de ataque: 25%.

El resto de formaciones y habilidades se encuentran en el anexo “Diseño”. También se puede leer en el documento otros modificadores de estadísticas como los niveles de las unidades.

Inteligencia artificial

Como se indica en el apartado anterior las unidades pueden ser usadas tanto por el jugador como por la IA. El jugador jugará y hará sus propias estrategias, pero la IA solo tendrá dos estrategias posibles con las unidades:

- **Comportamiento defensivo:**
  La unidad que tenga este comportamiento no se moverá de la casilla en la que reside. Si un enemigo se acerca lo suficiente (entra en su rango de alcance) le ataca. Será el comportamiento predominante entre los arqueros ya que al tener mas rango que el resto de unidades pueden atacar desde posiciones ventajosas (cuarteles, montañas...).

- **Comportamiento ofensivo**
  Una unidad con este comportamiento observará si hay unidades enemigas cercanas (dentro de su rango de movimiento). Si no la hay unidades enemigas mantiene la posición. Si encuentra unidades enemigas, elegirá una y se acercará (introduciendola en su radio de alcance) y la atacará.

4.3 Acciones por turno

Las unidades tendrán 6 acciones distintas pero no se podrán ejecutar todas en un turno. Solo se podrá realizar cada acción una vez por turno, además usar algunas acciones impide la utilización de otras. Si una unidad no puede usar más acciones ese turno se indica que esa unidad está usada.

Sanar: es una acción que solo se puede realizar si no se ha usado ninguna otra este turno. Esta acción hace que la unidad pase a estar usada. Esto hace que Sanar sea la más restrictiva de las acciones ya que usarla anula al resto y no permite usar otras acciones antes. Para poder sanar la unidad no debe tener ninguna unidad enemiga en las casillas contiguas.

Esperar: como sanar, anula al resto de acciones pero esta acción se puede usar si antes solo se ha usado una de las acciones restantes (mover, atacar, habilidad o cambio de formación). Siempre que se use esta acción la unidad pasa a estar usada.

Las 4 acciones restantes se pueden dividir en 2 grupos el de Movimiento (mover y cambio de formación) y el de ataque (atacar y habilidad). Realizar una acción de uno de los grupos anula a la otra acción de su grupo.

Mover: se podrá realizar si no se ha usado la otra acción de su grupo (formación). Tampoco se podrá usar si la unidad ya esta usada. Deberá existir casillas a las que se pueda mover.

Atacar: se podrá realizar si no se ha usado la otra acción de su grupo (habilidad). Tampoco se podrá usar si la unidad ya esta usada. Se necesita una unidad enemiga dentro del rango de ataque.

Cambio de formación: se podrá realizar si no se ha usado la otra acción de su grupo (mover). Tampoco se podrá usar si la unidad ya esta usada.

Habilidad: se podrá realizar si no se ha usado la otra acción de su grupo (atacar). Tampoco se podrá usar si la unidad ya esta usada. Se necesita una unidad enemiga dentro del rango de la habilidad.

Cuando una unidad usa una acción de cada grupo (movimiento o ataque) la unidad pasa a estar usada.

Todas estas limitaciones hacen que cada unidad solo pueda realizar como máximo 2 acciones por turno, obligando al usuario a gestionar sus movimientos.

4.4 Combate, visión, puntos por turno

La interacción de las unidades se dará mediante el combate y su capacidad de ver a otras unidades. Tras estos combates las unidades sufrirán daños que sanarán gastando puntos que recibirán tanto por combate como por el paso de los turnos.

Combate:
Cuando dos unidades se enzarzan en combate se harán dos cálculos. Estos cálculos son el daño recibido por las dos unidades enfrentadas.

El valor de “daño base de unidad” es la suma de los valores de cada soldado de la unidad, por lo tanto una unidad que tiene 20 hombres teóricamente tendrá el doble de ataque que una de 10 (si las dos unidades son de la misma clase).

En los cálculos de ataque o defensa de una unidad, también participarán los bonus de las casillas sobre las que se ubiquen, el cansancio de la unidad y si es de noche. Todos los

cálculos además de los datos sobre la visión y los puntos por turno se encuentran en el anexo “Diseño”

4.5 Condiciones de victoria y derrota

Para finalizar un tutorial el jugador deberá cumplir un objetivo determinado. Este objetivo de victoria podrá variar dependiendo del nivel. Podrán existir distintas posibilidades:

- **Destrucción del enemigo:**
  El jugador gana la partida eliminando a todas las unidades del bando enemigo.

- **Captura de una plaza:**
  El jugador podrá ganar si captura una posición determinada. Esta posición puede ser cualquier punto estratégico (cuarteles, ciudades, puentes...).

- **Soportar un avance enemigo:**
  Otra posibilidad es que el jugador deba defender una posición durante múltiples oleadas de enemigos durante un número determinado de turnos. Si el jugador repele el ataque gana.

El jugador perderá la partida si se cumple una condición de derrota. Esta puede ser:

- **Aniquilación del ejército aliado:**
  El jugador pierde todas las unidades de las que dispone

- **No cumple objetivo a tiempo:**
  El jugador no consigue la victoria en la cantidad de turnos indicados.

4.6 Historia en el juego

Aunque en este juego no hay una historia, los distintos tutoriales y nivel final se le dará una pequeña narrativa al usuario para darle un sentido al mapa que jugará. Esta narrativa se le dará a conocer al usuario mediante un interlocutor que le indicará que debe hacer (ver diseño de interfaz para saber la disposición del interlocutor en la pantalla). La narrativa no llegará más lejos que cosas como “el enemigo ha tomado X ciudad, debes retomarla” o “Nuestras fuerzas desembarcaron en sus playas, tome la ciudad Y para establecer una cabeza de playa”. Estas indicaciones vendrán acompañadas con algo de vocabulario militar para dar más empaque y contexto a la historia contada.

4.7 Justificación unidades

Como ya se ha dicho el juego no tiene historia como tal, pero he basado las unidades en cómo funcionaban (simplificando mucho) tropas de la época clásica (Romanas o Griegas). Esto no quiere decir que este juego se quede estancado y solo se podría diseñar una

historia teniendo en mente esa época. Lo único que habría que hacer para adaptarlo sería diseñar nuevas unidades y darle valores distintos a los que vienen a continuación o rediseñar las unidades actuales. El resto de comportamientos y mecánicas deberían funcionar de la misma forma.

Recordemos que el producto final de este TFG es un conjunto de mecánicas e interfaz para usarlas, las unidades y sus valores pueden variar para adaptarlo a las necesidades narrativas o estéticas.

5. Implementación

En esta sección se dará una explicación sobre la herramienta principal a la hora de desarrollar un videojuego: el motor de videojuegos. Se explicará como se ha seleccionado esta herramienta y se indicará que otras herramientas se usarán para el desarrollo del proyecto. Además se mostrará cómo se estructuran y gestionan los distintos archivos usados durante el desarrollo del proyecto dentro del motor seleccionado.

5.1 Motor de videojuegos

Un motor de videojuegos es una herramienta que ayuda a los desarrolladores a crear videojuegos sin tener que programar miles de líneas de código para crear “físicas” (como la gravedad y el rozamiento entre superficies). También dan sistemas de colision para saber cuando dos objetos colisionan y entornos de desarrollo tanto 3D como 2D para adaptarse a las necesidades de los desarrolladores. En la actualidad son herramientas indispensables para crear videojuegos por ello existen múltiples motores de videojuegos disponibles.

Como este proyecto no tendrá grandes necesidades gráficas o técnicas, características como “servicios en la nube” o “compatibilidad con tecnologías VR” no nos servirá para seleccionar el motor. Se elegirá el motor teniendo en cuenta otros factores menos técnicos, esto se debe a que el proyecto tiene 0 presupuesto, no se tiene documentación, falta de conocimiento en las herramientas y se requiere poder programar.

Factores usados para elegir el motor de videojuegos:

- **Precio:**
  Las distintas empresas emplean distintos sistemas de pago para poder usar sus motores. Como no se dispone de presupuesto se buscará una herramienta que se pueda usar de forma gratuita o sistemas que ofrezcan realizar el TFG de forma gratuita.

- **Capacidades de programación:**
  Existen motores preparados para desarrolladores que no saben programar como Stencyl [16], estos motores te dan ya creadas las mecánicas que puedes usar. La forma de hacer videojuegos con estas herramientas es como montar un puzle, no se programa, es un *Drag and Drop* de las piezas que te da el motor. Como el objetivo de este TFG es la realización de mecánicas, los motores que no den libertad a la hora de programar deben ser descartados.

- **¿Quién lo usa?:**
  Si el motor es solo usado por desarrolladoras de gran calibre o videojuegos de fama puede que no sea una herramienta válida para el TFG, por ejemplo el motor CryEngine [17]. Ya que la falta de experiencia en el medio y en las herramientas puede hacer que fracase. Se debe elegir una herramienta que sea usada por desarrolladoras pequeñas (las famosas empresas indie), como por ejemplo usan

RPGMaker. Este fue usado por la compañía indie *Freebird Games* para el desarrollo del juego “To the Moon” [18].

- **Otras utilidades:**
  Los motores de videojuegos son herramientas de grandes capacidades. Usarlas solo para realizar videojuegos es infrautilizarlas. Algunas compañías usan estas herramientas para otros fines como realidad aumentada, arquitectura y diseño de interiores... Por lo tanto aprender a usar estas herramientas podría ser beneficioso más allá del desarrollo del TFG.

- **Documentación:**
  El desarrollo que se va a realizar dependerá en gran medida de tutoriales y documentación que ayuden a la hora de desarrollar las distintas partes del proyecto. Por lo tanto necesitaré documentación extensa y gratuita.

Existen múltiples motores de videojuegos, a continuación se muestra una tabla comparando alguno de los más famosos y utilizados. En dicha tabla se indica si cumplen o no los factores explicados con antelación. El motor que mas apartados cumpla será el elegido.

	Motores					
Factores	**RPG maker**	**Stencyl**	**Unreal**	**Unity**	**Cryengine**	**GameMaker**
Gratis	✔	✔	✔	✔	✖	✖
Programable	✖	✖	✔	✔	✔	✔
Usado por indies	✔	✔	✔	✔	✖	✔
Otras utilidades	✖	✖	✔	✔	✔	✖
Documentación	✔	✔	✔	✔	✖	✔

Tabla 19 : *comparativa de algunos motores de videojuegos*

Como se puede ver en la tabla tanto Unreal como Unity son dos opciones viables para realizar el proyecto. La decisión entre estos dos es más personal que técnica. Aunque estos 2 motores son muy buenos, hay razones para pensar que Unity es más accesible. Por ejemplo, en el edificio Thinktic de Logroño han ofrecido y ofrecen cursos de esta herramienta. Por lo tanto para realizar este trabajo se usará el motor de videojuegos Unity de la empresa Unity Technologies.

5.2 Herramientas de trabajo

Ya seleccionada la herramienta principal para desarrollar el proyecto el resto de herramientas serán:

- Visual studio 2017: los juegos en Unity se programan en C#. Además visual studio tiene un plugin que permiten conectar su debugger con la plataforma de Unity, lo que permite probar el juego mientras usas el debugger.
- Blender: para la creación de elementos 3D que no se puedan realizar con Unity.
- GIMP 2: para la creación y modificación de imágenes.
- Cobian backup 11 Gravity: será usado para crear las copias de seguridad del trabajo.
- Drive (Google): como almacén y zona de trabajo de la documentación.
    - Hojas de cálculo de google.
    - Documentos de google.
    - Draw.i.
    - Presentaciones de google.
    - Diferentes complementos para mejorar la calidad de los documentos.
- Trello: será usado para gestionar las tareas a realizar en los diferentes sprints.
- Justinmind prototyper: para la creación de diseños de interfaz.

5.3 Estructura del proyecto

Como ya se ha dicho el motor de videojuego usado para realizar este proyecto será Unity. Para Unity todo elemento que el programador cree será un *Asset*, por eso todos los elementos estarán en la carpeta "*Assets*", ver imagen 17.

Los distintos tutoriales, niveles y menús los organiza en escenas (*scenes*). Todos los elementos existentes en una escena son un *gameObject* al que se le pueden añadir *gameObjects* hijos, comportamientos (*scripts*), efectos de audio, animaciones… y se pueden configurar sus características (posición espacial, nombre, *tags*...). Para poder trabajar con ellos, Unity permite almacenar una copia de un *gameObject* que has creado, esta copia se llama *prefab*. Los *prefabs* son el molde con el que podrás crear múltiples *gameObjects* en la escena. El objetivo de esto es que el programador cree un *gameObject*, lo convierte en *prefab* y luego use el *prefab* para crear múltiples copias con las que se trabajara.

El resto de material audiovisual (imágenes, audios, materiales...) y código (*scripts*) también serán *assets*. Esto hace que el programador sea el que organice los *assets* como vea conveniente.

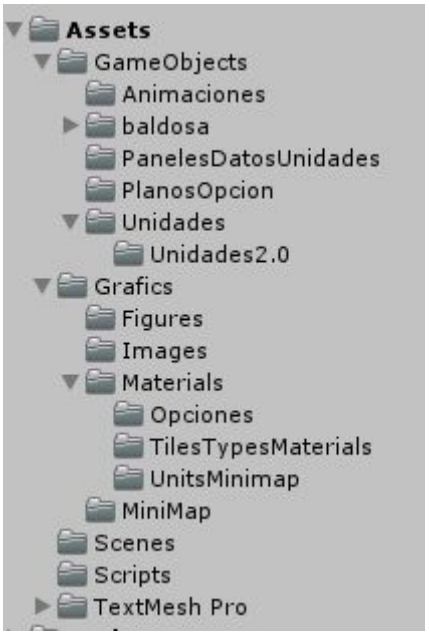


Imagen 17: *estructura de la carpeta “Assets”*

6 Incrementos

Esta sección trata el desarrollo de los distintos *sprint* (incrementos) que se realizaron en el proyecto. Cada apartado explicará:

- El objetivo del *sprint*.
- Los apartados más destacables de la implementación que se diera en ese *sprint.* La totalidad de la implementación se encuentra en el anexo “Sprints”.
- Problemas más reseñables que se dieron durante la implementación del *sprint.*
- 3 Gráficos que explican cómo se desarrolló el *sprint* respecto a lo planificado.
    - Tabla donde se muestra cada tarea con las horas planificadas para la misma y el consumo real de horas.
    - Gráfico Gantt donde se puede ver la distribución semanal de las tareas según la planificación inicial y la distribución en la que se realizó.
    - Gráfico *Burn up* que compara (según las horas planificadas) como se desarrollaron las actividades contra la planificación.
- La información más destacable que se extrajo de las encuestas realizadas a los testeadores.

6.1 Primer incremento

El primer *sprint* tiene como principal objetivo implementar las mecánicas más básicas de un juego de estas características (distintos tipos de unidades, como se mueven, el terreno...), además se usará como aprendizaje de la herramienta Unity, ver tabla 20.

		SEMANAS		
Primer Sprint	**Horas estimadas**	**S1**	**S2**	**S3**
Mapa	4			
Tipos de suelo	4			
Movimiento (posibles movimientos)	8			
Movimiento (Camino más corto)	8			
Unidades aliadas	4			
IU minimapa mensajes y menús básicos	8			
Pantalla de Inicio	6			
Primer tutorial	8			
Total:	50			

Tabla 20: *diagrama de gantt del primer sprint*

Desarrollo destacado

Movimiento

Para explicar cómo una unidad sabe a qué casillas se puede mover primero hay que explicar el concepto de “casillas vecinas”. Estas “casillas vecinas” son todas las casillas que rodean a una casilla, su número puede variar entre 1 y 6 dependiendo de la posición de la casilla. La casilla que tiene una unidad encima envía 6 “rayos” uno por cada uno de sus aristas, si estos rayos chocan contra otra casilla se registra esa casilla como vecina. Lo importante aquí es saber que estos rayos se envían siempre en un orden concreto. En la imagen 18 se ve el orden de estos rayos.

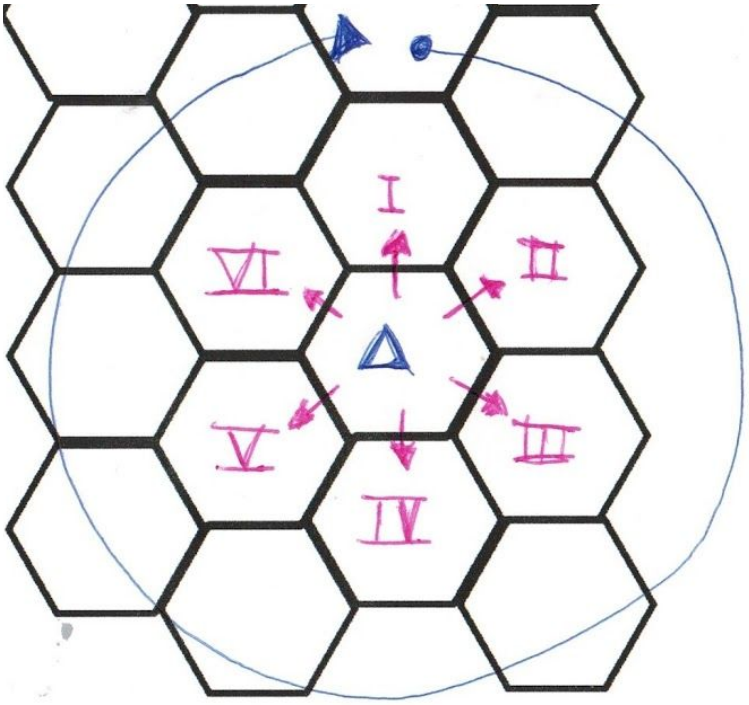


Imagen 18: *esquema del orden de los rayos enviados.*

A esto hay que añadir 4 factores para saber si una casilla es apta para mover la unidad a ella.

- Capacidad de movimiento de la unidad: lo que es capaz de moverse la unidad.
- Coste de movimiento de la casilla: lo que cuesta desplazarse a la casilla.
- Valor máximo de llegada: es la capacidad máxima de movimiento que tiene la unidad al llegar a esa casilla, se usa para calcular a qué casillas vecinas se puede ir.
- Casilla ocupada: si la casilla está ocupada por otra unidad, la casilla no es apta para moverse.

Una unidad para desplazarse observa con qué capacidad de movimiento llega a todas las vecinas de su casilla. Si esa capacidad de movimiento es mayor que el coste de movimiento de la casilla, la unidad se podrá desplazar a esa casilla. Desde esa casilla se calculará a que vecinas se puede desplazar usando para ello el “valor máximo de llegada”. Si una casilla aun habiendo calculado a que vecinas puede llegar recibe un “valor máximo de llegada” mayor al que había usado para calcular las vecinas, vuelve a calcular todo.

Con este proceso se asegura que una unidad podrá llegar a todos los puntos del mapa desde cualquier ángulo siempre y cuando quede algo de capacidad de movimiento. Para hacerlo más visual puede verse la imagen 19 donde se explica de forma gráfica estos párrafos. En la imagen se muestra la unidad representada como un triángulo azul, las casillas a las que la unidad se puede desplazar con un tick verde. En todas las casillas se muestra el valor máximo de llegada y el coste de cada casilla. Como obstáculo se ha colocado una unidad enemiga (triangulo rojo).

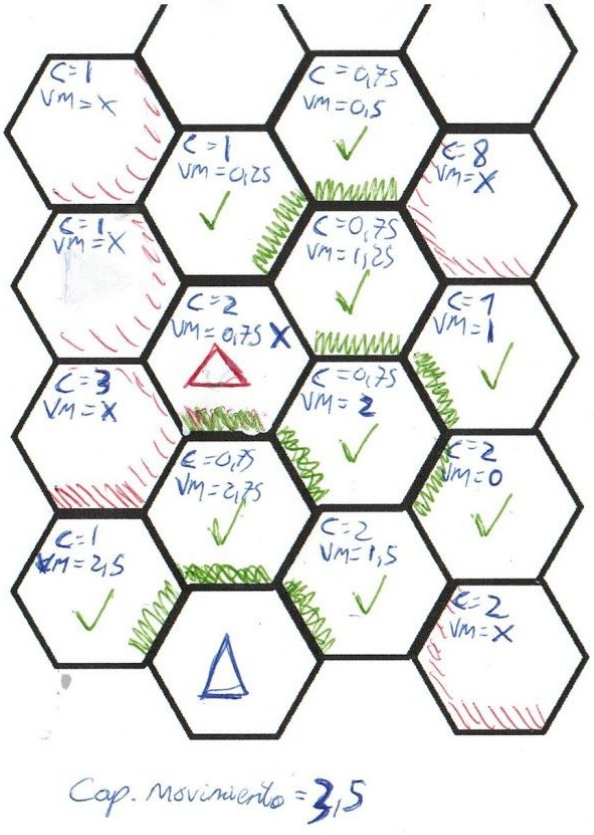


Imagen 19: *ejemplo de movimiento de una unidad*

Camino más corto:

Para evitar que nuestras unidades al desplazarse ejecuten movimientos extraños (como que atraviesen ríos o pasen por encima de unidades enemigas). El desplazamiento de las mismas se realizará siguiendo un camino de casillas “despejadas”. Además la unidad realizará la ruta más corta hasta la zona indicada. Se quiso implementar un algoritmo como Manhattan (que no se puede ya que las casillas no son cuadradas) o Dijkstra (que requiere pre-calcular el grafo de todas las casillas). Al final no se consiguió realizar y se realizó un método que resuelve este problema de forma menos elegante pero que es dinámico (por lo que podría cambiar el mapa durante la partida y no afectaría al programa) y no necesita almacenar nada en memoria.

El método consiste en usar las casillas “despejadas”, que son las casillas que se habían indicado como aptas en el movimiento. El camino más corto puede ser un camino distinto al

que se ha usado para llegar a la casilla objetivo con los valores de movimiento (el camino que optimiza los valores mayores de llegada). Esto se debe a que para calcular el camino más corto se usa todas las casillas aptas y su cercanía al objetivo. Esto quiere decir que para elegir a qué casilla se moverá la unidad se pregunta a las casillas vecinas despejadas cual es la más cercana (se usan los centros de las casillas para calcular esta distancia) al objetivo y se añadirá a una lista que será el camino que usará la unidad como “camino más corto”, luego se seguirá un ciclo iterativo hasta que se llegue a la casilla objetivo. Siguiendo el ejemplo del camino más corto en la imagen 20 se ve cuál sería el camino más corto hasta la casilla objetivo indicado por las flechas rosas.

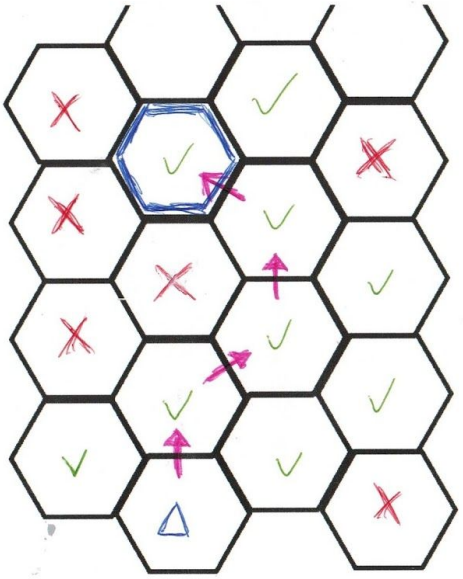


Imagen 20: camino que seguirá la unidad (triángulo azul) hasta el objetivo (casilla con reborde azul)

Problemas

Durante este sprint gran parte de los problemas se debieron a que fue el primer contacto con las herramientas de desarrollo.

Diseño, Planificación, Análisis + sprint

La cantidad de horas que he invertido en el periodo de tiempo del primer sprint ha hecho que alguna tarea se retrasara. La solución ha sido que usar parte de la 4 semana para hacer tareas de la 3 semana.

Movimiento (grafos)

Tras mucho leer, investigar y probar el no conseguir hacer un sistema de grafos que funcione en las casillas de mi juego mé ha hecho perder mucho tiempo. La solución ha sido crear métodos que no usen grafos para calcular las distancias y recorridos.

Problemas en el tutorial

Tuve problemas a la hora de crear el tutorial, me ocurrieron distintos fallos que eran completamente nuevos, lo que me hizo perder mucho tiempo arreglarlos. Además, poco antes de enviar el tutorial a los testeadores encontré unos fallos mecánicos que quise arreglar antes de enviarlo. La idea de los testeos es que encuentren fallos nuevos no que encuentren cosas que ya he visto yo.

Gráficos de seguimiento

Algunas tareas como movimiento fueron mucho más complicadas de realizar de lo pensado. Pero otras fueron mucho más simples. Aunque el cómputo de horas no dista de las horas objetivo algunas tareas varían mucho entre lo planificado y lo realizado, ver tabla 21.

Horas planificadas y horas realizadas

Primer Sprint	Horas estimadas	Horas Realizadas
Mapa	4	2
Tipos de suelo	4	2
Movimiento (posibles movimientos)	8	14
Movimiento (Camino mas corto)	8	6
Unidades aliadas	4	4
IU: minimapa mensajes y menús básicos	8	8
Pantalla de Inicio	4	1
Primer tutorial	8	10
	48	**47**

Tabla 21: *horas estimadas y horas realizadas (sprint 1)*

El realizar el primer *sprint* junto a la planificación, diseño y análisis ha hecho que la tarea del tutorial se retrasara una semana. Mientras que la realización de las unidades aliadas se adelantó para trabajar mejor en las tareas de movimiento, ver tabla 22.

	SEMANAS			
Primer Sprint	**S1**	**S2**	**S3**	**S4**
Mapa				
Tipos de suelo				
Movimiento (posibles moviminentos)				
Movimiento (Camino mas corto)				
Unidades aliadas				
IU minimapa mensajes u menús básicos				
Pantalla de Inicio				
Primer tutorial				

Tabla 22: *diagrama de gantt comparativo (sprint 1)*

En el gráfico *burn up* se puede ver claramente que aunque se empezó con un adelanto sobre la planificación, se acabó retrasando tareas hasta la 3ª semana, ver imagen 21.

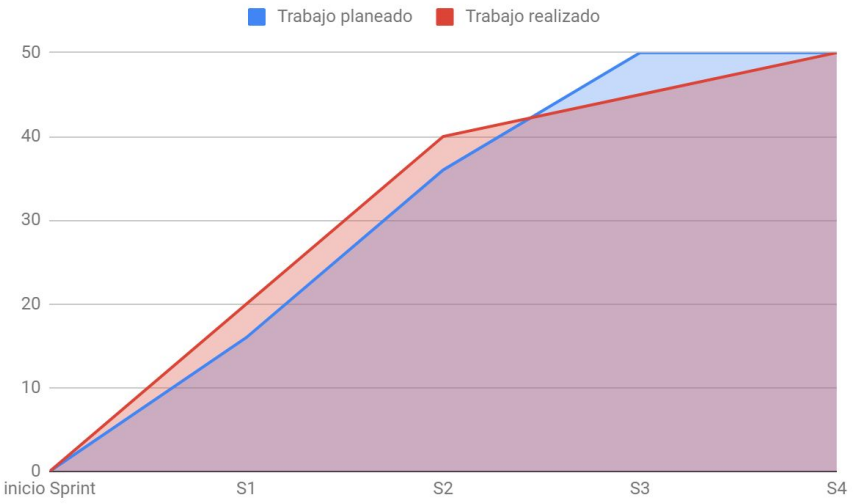


Imagen 21: *gráfico burn up (sprint 1)*

Testeos

Los testeadores han probado el prototipo en sus respectivos ordenadores (de diferentes gamas) desde un Intel core viejo sin tarjeta gráfica dedicada hasta un ordenador con los mejores componentes del mercado. Del formulario que rellenaron se pudieron extraer las siguiente conclusiones:

Errores de mecánicas:

- Ninguno encontró fallos mecánicos (objetivo principal del testeo)
- Menos el intel antiguo, ningún usuario ha tenido problemas de rendimiento.

De lo dicho y propuesto por los usuarios solo tomaré dos detalles para modificar:

- El minimapa debería mostrar más el campo que ves delante y no mostrar tanto el de detrás.
- Colocar un fondo de color y no dejar el fondo por defecto.

6.2 Segundo incremento

Este sprint tiene como principal objetivo el desarrollo de las primeras mecánicas pertenecientes al sistema de combate del juego, ver tabla 23.

		SEMANAS		
Segundo Sprint	**Horas estimadas**	**S1**	**S2**	**S3**
Reparar bugs del testing	4			
Unidades Enemigas	4			
Mecánica de turnos (solo que pasen turnos)	8			
Ataque el menú de acción básico	8			
IA (básica)	8			
IU (Datos de unidades y terreno)	4			
Segundo Tutorial	4			
Total:	40			

Tabla 23: diagrama de gantt del segundo sprint

Desarrollo destacado

Menú de acción básico (espera, movimiento y ataque):

En el anterior sprint la única interacción con las unidades era la posibilidad de mover las tropas. Ahora las tropas pueden hacer 3 acciones. Para elegir entre estas acciones, el usuario debe pulsar sobre una unidad, a su alrededor aparecerán 3 cuadrados (ver imagen 22), cada uno representará una acción. Si se pulsa sobre una de ellas la acción pulsada realizará su función:

Movimiento (MOV): no varía con respecto al anterior sprint, se iluminan las casillas alcanzables y si el usuario pulsa en una la unidad se desplazará a ella.

Ataque (ATA): las unidades enemigas (dentro del alcance) se iluminan y si una de estas es seleccionada combatirá contra tu unidad.

Esperar (ESP): la unidad no hace nada y consume el resto de acciones. En el siguiente sprint se le dará mayor funcionalidad.

Cada acción tendrá un color distinto e identificativo. Si una de las acciones anteriores no se puede realizar (ver documento de especificación de las mecánicas de estas acciones) el color de la casilla será gris para indicar que la acción no se puede realiza,ver imagen 22.

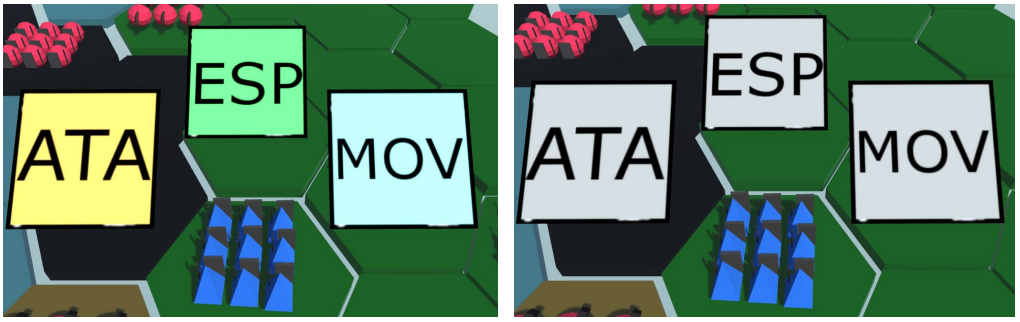


Imagen 22: *captura donde se ve las acciones disponibles (con colores) y usadas (gris)*

IA:

Se ha programado una IA muy sencilla que interactuara con el usuario moviendo al bando enemigo. Con ello se pretende que el jugador pueda jugar solo contra el PC. Las unidades de la IA tendrán dos comportamientos posibles:

Defensivo: la unidad se mantiene en una posición constante y la defiende atacando a cualquier unidad que entre en su radio de ataque.

Ofensivo: la unidad comprobará si existen unidades enemigas dentro de su alcance de movimiento. En caso afirmativo se desplazará a una casilla contigua y atacará a la unidad, ver imagen 23.

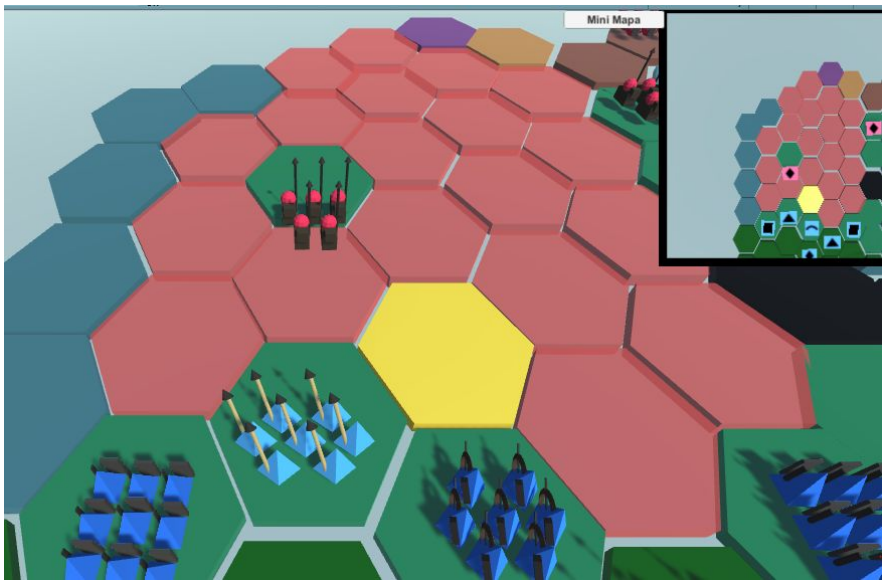


Imagen 23: *captura realizada mientras una unidad es desplazada por la IA. La casilla amarilla es la casilla a la cual la unidad se desplaza.*

Problemas

A pesar de no ser ya nuevo con Unity siguieron surgiendo fallos y complicaciones. Además todavía no comprendía la totalidad de las herramientas que aporta Unity, por lo que el desarrollo en algunas partes fue complicado.

IA

Programar una IA básica ha sido más complicado de lo esperado. Fue complicado hacer unos comportamientos básicos (solo defender o solo atacar). Se quería haber puesto otro comportamiento pero la falta de tiempo impidió esto. Se llegó a la conclusión de que invertir más tiempo en la IA sería bastante perjudicial para el objetivo del proyecto (implementar múltiples mecánicas).

Re-Planificación

Durante la implementación se vio que sería necesario hacer mejoras en la interfaz, además los testeadores reflejaron que algunos apartados visuales necesitaban un repaso. Esto sumado a que mejorar la IA parecía ser una mala inversión del tiempo y que algunas mecánicas planteadas no parecían adecuadas. Produjo que se realizará una segunda planificación de los sprint restantes.

Incompatibilidad entre tutoriales

Al realizar el segundo tutorial se descubrió que no era posible tener 2 o mas tutoriales simultáneamente en el proyecto. Este problema no se debe a la herramienta (Unity), el causante es la diferencia de versión de los objetos. Al modificar las características de un componente (mejorarlos para que tenga más funcionalidad) también se cambiará en todos los tutoriales ya que el objeto prefabricado es el mismo.

Por ejemplo, en el tutorial 1 las unidades solo se podían mover, en el tutorial tienen 3 acciones. Esto hará que al volver a compilar el tutorial 1 las unidades adquirieron 3 acciones lo que romperá lo explicado en el primer tutorial. Básicamente no se pueden mantener 2 versiones de un mismo objeto a no ser que generes 2 versiones distintas del mismo objeto.

Esto repercute en que según el diseño, desde la pantalla de inicio se podría seleccionar entre 3 tutoriales y un nivel. Pero solo se podrá 1 por versión (cada sprint genera una nueva versión)

Gráficos de seguimiento

En el segundo *sprint* se predijo mejor la duración de las tareas. Por lo tanto las horas que costó realizar las actividades no se alejaron de las horas planificadas,ver tabla 24.

SegundoSprint	Horas estimadas	Horas Realizadas
Reparar bugs del testing	4	2
Unidades enemigas	4	3
Mecánica de turnos (solo que pasen turnos)	8	7
Ataque y el menú de acción básico	8	9
IA (básica)	8	10
HUB (Datos de unidades y terreno)	4	2,5
Segundo Tutorial	4	6
	40	**39,5**

Tabla 24: *horas estimadas y horas realizadas (sprint 2)*

Con respecto al orden y organización a lo largo de las semanas se cumplio la planificación a rajatabla, ver tabla 25.

	SEMANAS		
SegundoSprint	S1	S2	S3
Reparar bugs del testing			
Unidades enemigas			
Mecánica de turnos (solo que pasen turnos)			
Ataque y el menú de acción básico			
IA (básica)			
HUB (Datos de unidades y terreno)			
Segundo Tutorial			

Tabla 25: *diagrama de gantt de actividades (sprint 2)*

En el gráfico *burn up* se ve como se cumplio a rajatabla la planificación, ver imagen 24.

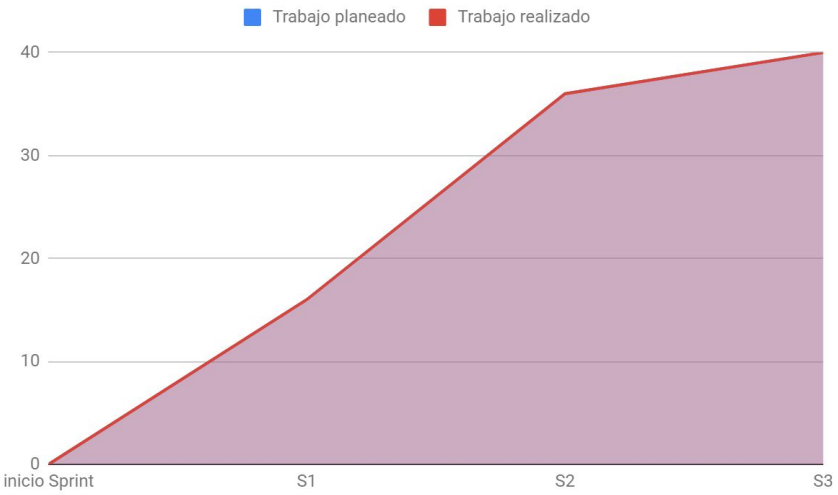


Imagen 24: *gráfico burn up (sprint 2)*

Testeos

La segunda remesa de tests hizo aflorar un fallo mecánico y 2 de diseño.

Error de solo el boton ESP
El único error mecánico que se recalco fue que a veces al pulsar sobre una unidad sucedía que solo se mostraba el cuadrado de acción de esperar (ESP). Este error sucedia por que la unidad no tenía registrada la casilla inferior y al intentar interactuar con ella (necesario para mover y atacar) creaba una excepción. Se arregló sin problemas.

Unidades muy poderosas
Los testeadores recalcaron que una unidad de caballería era muy poderosa mientras que los escuderos eran muy débiles. Además el cálculo de daños en un combate no era muy equilibrado. Todo esto planteó la necesidad de crear una tarea para balancear a las unidades.

Control cámara:
Un testeador recalcó que sería necesario cambiar los botones de control de la cámara, la cruceta no le parecía conveniente y pedía que fueran los botones (“A,S,D,W”). Esto hizo pensar que sería necesario preguntar a los jugadores que controles creen que faltan.

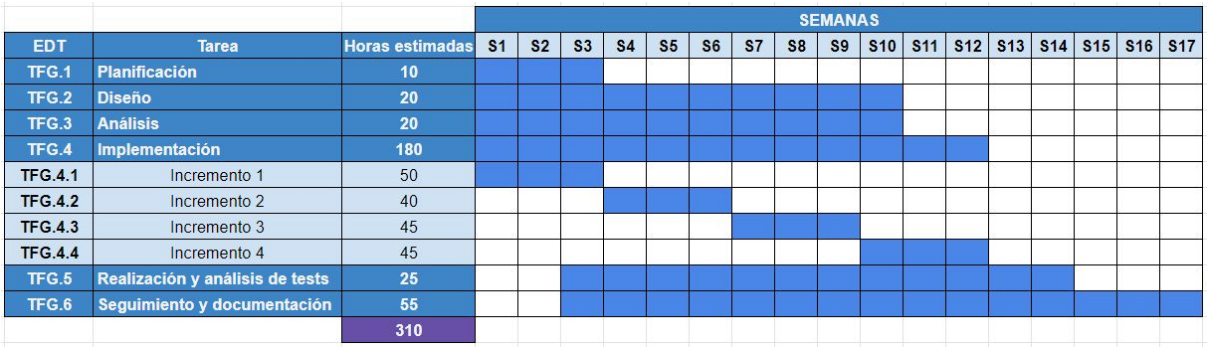
6.3 Re-planificación

En este punto del trabajo se vio que habían surgido unas necesidades que no se podían solventar con la planificación existente. Se optó por realizar un reajuste en la planificación que se adaptara mejor a las necesidades. Todo el reajuste se puede leer en el anexo “Re-planificación”, a continuación se muestra el gantt modificado, ver imagen 25.

Nuevo gantt

El gantt obtiene muchas modificaciones respecto a su versión inicial.

- Se amplía el tiempo de diseño y análisis ya que en él a lo largo de todos los sprints se realiza pequeños reajustes sobre el diseño inicial, esto no se había contabilizado en el primer *Gantt*.
- Se elimina un *sprint* y se reajusta el volumen de horas de los sprints 3 y 4.
- Se amplían las horas a la tarea de seguimiento y documentación.



			SEMANAS															
EDT	Tarea	Horas estimadas	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17
TFG.1	Planificación	10																	
TFG.2	Diseño	20																	
TFG.3	Análisis	20																	
TFG.4	Implementación	180																	
TFG.4.1	Incremento 1	50																	
TFG.4.2	Incremento 2	40																	
TFG.4.3	Incremento 3	45																	
TFG.4.4	Incremento 4	45																	
TFG.5	Realización y análisis de tests	25																	
TFG.6	Seguimiento y documentación	55																	
		310																	

Imagen 25: *nuevo gráfico Gantt*

Los sprints explicados a continuación se realizaron siguiendo la nueva planificación.

6.4 Tercer incremento

Este *sprint* tenía como objetivo mejorar el combate presentado en el anterior sprint. Para ello se balanceo las características de las unidades (daño, defensa, salud) y se les añadió múltiples mecánicas que aumentaran las posibilidades del combate, ver tabla 26.

		SEMANAS		
Tercer Sprint	Horas estimadas	S1	S2	S3
Reparar bugs del testing	4			
Balancear tropas	4			
Niveles de las unidades	8			
Habilidades y formaciones	8			
Turnos avanzados (ciclo dia noche)	8			
Sanar	8			
Tercer tutorial	4			
Total:	44			

Tabla 26: diagrama de *gantt del tercer sprint*

Desarrollo destacado

Piedra - papel - tijera:

Para equilibrar más el combate se añadieron unas instrucciones al sistema de cálculo de daños que modifica el daño dependiendo del tipo de unidades que se enfrentaban. Esto se hace para crear un sistema que se suele denominar “piedra - papel - tijera”, ver imagen 26.

Esta modificación hace que:

- Los caballeros son buenos contra escuderos
- Los escuderos son buenos contra los lanceros
- Los lanceros son buenos contra los caballeros
- Los arqueros no son buenos contra nadie pero no reciben daño en los combates en los que es el atacante.

Si una unidad se enfrenta a una unidad contra la que no tiene ventaja recibirá una penalización en el daño que hace y la unidad contraria hará mucho más daño.

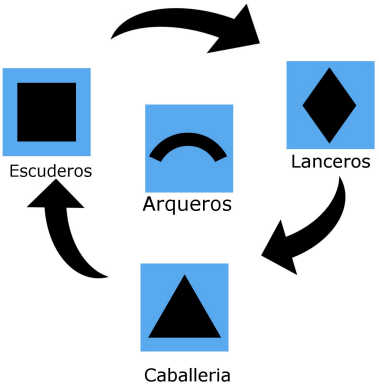
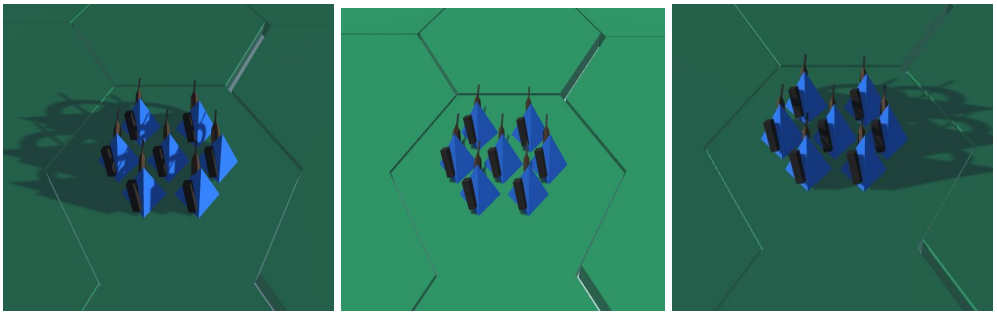


Imagen 26: *esquema de debilidades*

Ciclo solar:

El transcurso de las rondas no es en valde. El tiempo pasa y por ello la disposición del sol varía, esto hace que la iluminación del campo de batalla cambie. Cada ronda representará 3 posibles posiciones del sol (mañana, tarde, noche) que se irán rotando, ver imagen 27. Cuando sea de noche las unidades (todas) recibirán una penalización de una reducción del 20% en el daño que hagan.



Imágen 27: *cambios de iluminación según la franja horaria (mañana, tarde y noche)*

Problemas

Durante este sprint gran parte de los problemas se debieron a que el código que había sido desarrollado en unas primeras fases era muy enrevesado. Se tuvo que invertir tiempo en reescribir algunas partes con los nuevos conocimientos adquiridos.

Habilidades y formaciones

Las nuevas acciones fueron más costosas de hacer y programar de lo esperado. Las habilidades requieren acciones especiales dentro de los algoritmos de ataque que hicieron necesario repasar mucho de su código.

Tercer tutorial

Ajustar las distintas mecánicas para que funcionasen correctamente costó más tiempo de lo esperado. Además construir un mapa de gran tamaño desempeñó problemas que se predijeron pero eran más grandes de lo esperado (mucho coste en la colocación de casillas de distintos tipos o la dificultad de crear muchos accidentes geográficos). En el siguiente sprint habrá que buscar una solución.

Gráficos de seguimiento

Tras la re-planificación en el tercer *sprint* se predijo la duración de las tareas bastante mal. La mayoría de actividades costaron menos tiempo realizarla que lo planeado pero la realización del tutorial consumió mucho más tiempo del pensado, ver tabla 27.

Tercer tutorual	Horas estimadas	Horas Realizadas
Reparar bugs del testing	4	2
Balancear tropas	4	4
Niveles de las unidades	8	4
Habilidades y formaciones	8	11
Turnos avanzados (ciclo dia noche)	8	5
Sanar	8	4
Tercer tutorial	4	8
	44	**38**

Tabla 27: *horas estimadas y horas realizadas (sprint 3)*

En este sprint se gozó de más tiempo durante la primera semana, que se aprovechó para adelantar trabajo. Como se puede ver en el tabla 28 se pudo adelantar mucha tarea.

	SEMANAS		
Tercer tutorual	**S1**	**S2**	**S3**
Reparar bugs del testing			
Balancear tropas			
Niveles de las unidades			
Habilidades y formaciones			
Turnos avanzados (ciclo dia noche)			
Sanar			
Tercer tutorial			

Tabla 28: *diagrama de gantt de actividades (sprint 3)*

En el gráfico *burn up* muestra claramente como las tareas se realizaron antes que lo planificado, ver imagen 28.

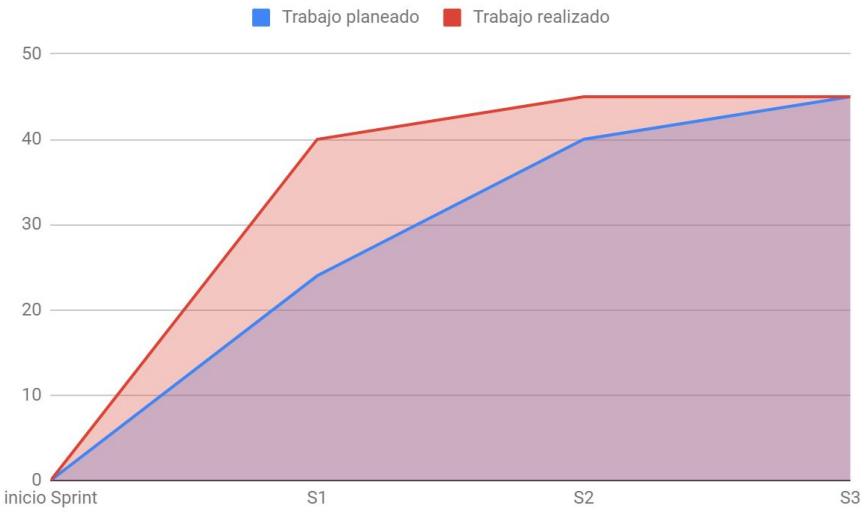


Imagen 28: *gráfico burn up (sprint 3)*

Testeos

No reportaron casi ningún error, la mayoría de los comentarios fueron peticiones y sugerencias visuales

Simulador atascado
El único error registrado es que el simulador a veces se queda bloqueado tras un combate (debería desaparecer tras el combate). Se ha resuelto añadiendo alguna mejora al código.

Cambios visuales

Los usuarios vieron la necesidad de hacer cambios visuales a gran escala para mejorar el entendimiento de las mecánicas. Para resolver estas necesidades durante la planificación se comentaran dichas necesidades en el *thinking aloud* para aglutinar las mejoras dichas por los usuarios.

Thinking aloud

En este punto del proyecto (entre el tercer y cuarto *sprint*) se realizó un *thinking aloud* en el que participaron tres testeadores. Esta experiencia fue muy productivo y se obtuvo mucha información útil. A continuación se muestran dos ejemplos de los temas tratados durante el *thinking aloud,* para más información ver el anexo “Thinking aloud”.

Barra de unidad:
Aunque la ventana de datos es interesante para los usuarios, opinan que a simple vista las unidades del tablero deberían dar más información. Indicaron que sería interesante tener algunos datos (Salud, experiencia y nivel) sobre las unidades del tablero. Para ello se pensó en el modelo de interfaz de una unidad que se usa en múltiples juegos. Por ejemplo los videojuegos de la saga Pokémon, ver imagen 29.

Imagen 29: *interfaz de datos que se ve durante un combate en el videojuego Pokémon versión verde hoja.*

Nuestra interfaz no requerirá del nombre y el género de la unidad por lo tanto se podría tener algo como lo que se muestra en la imagen 30.

Imagen 30: *posible interfaz de datos sobre las unidades.*

El nivel indicaría a que nivel se encuentra la unidad con un número. En la barra de salud se mostraría no un número sino una “barra de carga” que se iría reduciendo según la unidad pierda puntos de salud, teniendo como máximo 25 puntos de salud. En la barra de experiencia se mostrará una “barra de carga” que se rellenará hasta alcanzar el siguiente nivel. Al alcanzar el nivel máximo esta barra podría desaparecer.

Cansancio:
El cansancio debería ser indicado de distinta forma. se propuso que se añadiera una barra con 3 segmentos pintados de un color cuando el cansancio es máximo y que se vayan vaciando los segmentos por cada turno sin descansar, ver imagen 31. También se dijo que la penalización por cada turno sin descansar debería de aumentar de 10% a 25% para que las unidades se vuelvan inútiles en solo 4 turnos no en 10 (10 turnos les resultó excesivo).

Imagen 31: *idea de representación del cansancio.*

6.4 Cuarto incremento

Este *sprint* tiene como principal objetivo introducir una gran cantidad de mejoras descritas en el *Thinking aloud* que se realizó en el anterior sprint.

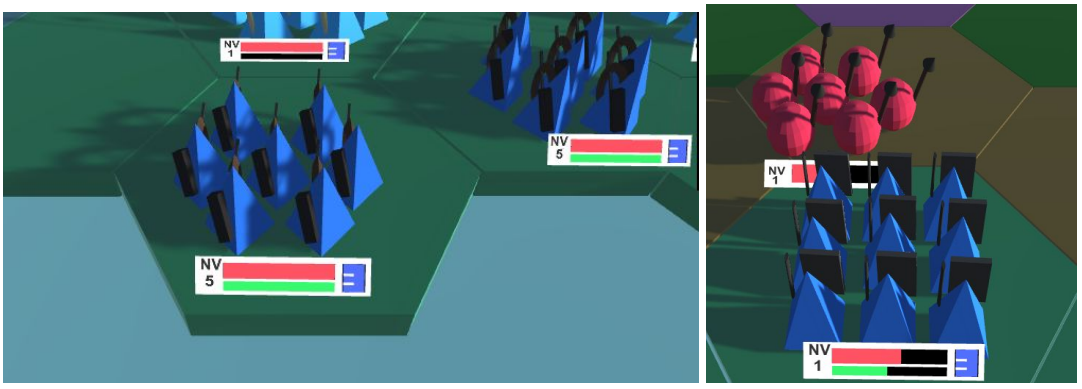
Además se añadieron otras 2 mecánicas que interfieren con el tablero más que con el jugador o las unidades, ver tabla 29.

		SEMANAS		
Cuarto Sprint	**Horas estimadas**	**S1**	**S2**	**S3**
Reparar bugs del testing	4			
Implementación Thinking aloud	10			
Spawns	4			
creación de bloques de mapa	6			
Mecánica de aparición de mapa	4			
Guardar partida	8			
Primer nivel	8			
Total:	44			

Tabla 29: diagrama de gantt del cuarto sprint

Desarrollo destacado

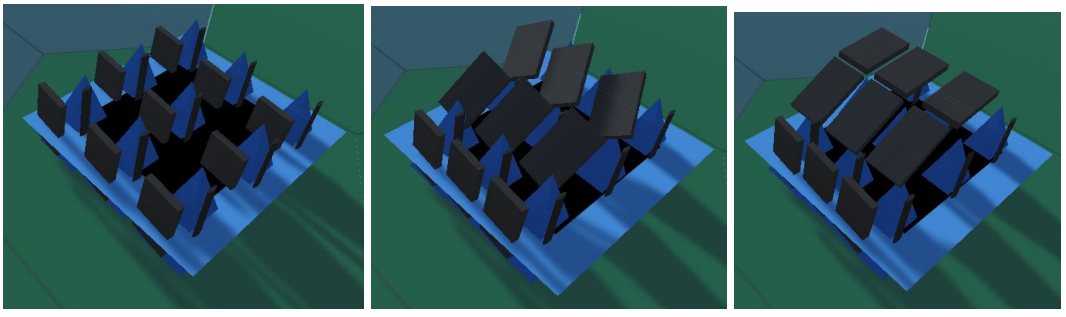
Barras de datos:
En el *thinking aloud*, se indicó que sería conveniente que las unidades tuvieran una pequeña ventana cerca que indicase mediante barras (la salud, la experiencia, el nivel y el cansancio), ver imagen 32. Esto se resolvió colocando a las unidades una ventana con tres barras (2 horizontales y una vertical) que indican la salud (rojo), la experiencia restante hasta el siguiente nivel (verde) y el cansancio de la unidad (azul). La parte consumida de estas barras será indicada por el color negro. El nivel está escrito a la izquierda de las barras. En el caso de las unidades enemigas solo tendrán una barra para la salud y el nivel al que se encuentra.



Imágen 32: *capturas del juego donde se muestran las ventanas de barras en las unidades*

Animaciones:
En el *thinking aloud*, se indicó que las unidades deberían tener una forma física distinta cuando se modifica la formación. Además se indicó que las unidades deberían de tener animaciones para darles más “vidilla”. Por lo tanto se añadieron 3 animaciones distintas a las unidades aliadas.

- Animación de reposo: la unidad está esperando una acción.
- Animación de movimiento: la unidad se desplaza por el tablero.
- Animación de formación: se activa para indicar que la unidad está en formación, ver imagen 33.



Imágen 33: *capturas tomadas durante la animación de la formación tortuga*

Problemas

Durante este sprint los problemas se debieron a que algunas tareas que estaban asignadas a fueron más complicadas de implementar de lo esperado.

Guardar
Se realizó la investigación pertinente para descubrir cómo guardar la partida y se descubrió que había que hacer demasiados cambios en el código. Además muchas de las mejoras descritas en el *thinking aloud* eran más beneficiosas al proyecto (ajuste de algunas mecánicas, mejoras visuales, mejoras de interfaz...) que guardar (que no es siquiera una mecánica) por lo que se decidió suprimir esta tarea y invertir más tiempo en el *thinking aloud.*

Nivel (Tutorial 4)
Las nuevas mecánicas (que afectan mucho al mapa) fueron costosas de implementar, pero lo que hizo que la tarea ocupase tanto tiempo fue el ajustar muchos parámetros y elementos del juego para evitar problemas con las mejoras que se implementaron del *thinking aloud.*

Gráficos de seguimiento

El cuarto sprint fue el más accidentado de todos con respecto a la planificación. Durante la primera semana estipulada no se pudo invertir tiempo en el TFG lo que produjo que todas las tareas se retrasara una semana. Además, viendo la dificultad de realizar la tarea "Guardar partida" y la cantidad de información conseguida del *thinking aloud* seria mejor para el proyecto desechar la primera e invertir más tiempo en las mejoras indicadas en la segunda. Todos estos cambios se reflejaron en la inversión de horas de las distintas tareas que variaron mucho con respecto a lo planificado, ver tabla 30.

Cuarto Tutorial	Horas estimadas	Horas Realizadas
Reparar bugs del testing	4	3
Implementación Thinking aloud	10	17
Spawns	4	4,5
creación de bloques de mapa	6	4,5
Mecánica de aparición de mapa	4	2
Guardar partida	8	2
Primer nivel	8	13
	44	**46**

Tabla 30: *horas estimadas y horas realizadas (sprint 4)*

Como se ha indicado todas las tareas sufrieron retrasos por lo que casi nada se realizó cuando estaba planificado realizarlo, ver tabla 31.

	SEMANAS			
Cuarto Tutorial	S1	S2	S3	S4
Reparar bugs del testing				
Impletmentacion Tihinkig aloud				
Spawns				
creacuion de bloques de mapa				
Mecanica de aparicion de mapa				
Guardar partida				
Primer nivel				

Tabla 31: *diagrama de gantt de actividades (sprint 4)*

En el gráfico burn up se puede ver claramente el retraso que sufrió todo el sprint con respecto al plan, ver imagen 34.

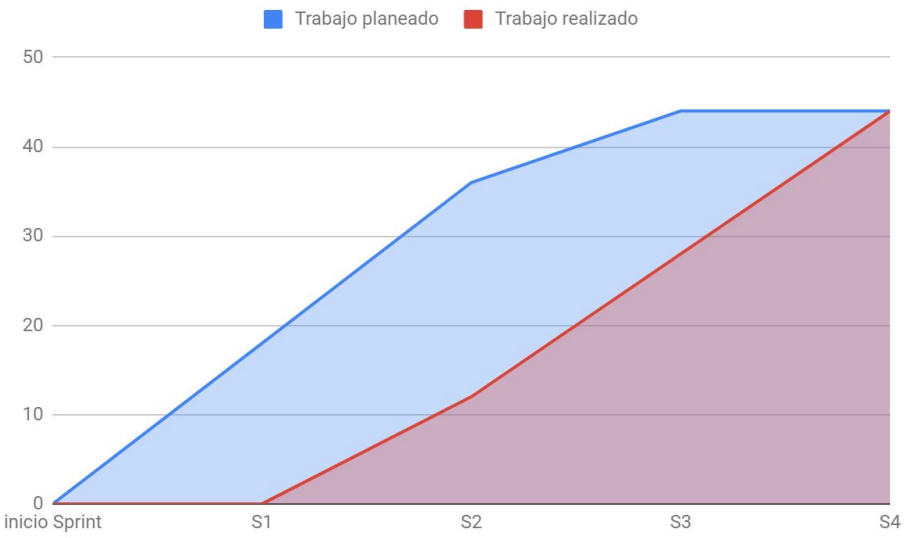


Imagen 34: *gráfico burn up (sprint 4)*

Testeos

Los test de este sprint fueron muy útiles para confirmar que las mejoras realizadas fueron bien recibidas por los testeadores.

El simulador:

Sigue quedándose atascado pero solo ha pasado una vez mientras que anteriormente pasaba mucho más frecuentemente. Las mejoras que se hicieron en el código fueron útiles pero no resolvieron el problema del todo.

Error visual:

Se ha informado de que hay un error visual en el minimapa. Los lanceros aliados no se muestran en ocasiones. Se ha podido concluir que el causante es el panel que está adjudicado a cada unidad (muestra la imagen de un rombo en el minimapa). Está tan a ras de suelo que a veces se pone por debajo del mismo. Esto genera el fallo.

Resumen de la implementación

En general la implementación se ha cumplido conforme a lo planificado. Los *sprints* se han realizado mayoritariamente dentro de las horas y fechas estipuladas. Prácticamente todas las tareas asignadas a los distintos sprint se finalizaron de forma satisfactoria. Además casi no hubo interrupciones durante el desarrollo, lo que ayudó a dejar tiempo al final para organizar y documentar todo lo desarrollado.

7.Conclusiones

En este apartado se dará una visión general de cómo ha sido el desarrollo del proyecto, como se podría mejorar y qué cosas he podido aprender al realizar este proyecto.

7.1 Balance

En líneas generales se ha cumplido con el alcance inicial aunque alguna cosa se ha quedado sin realizar en pos de mejorar otras. En el apartado técnico el proyecto funciona de forma adecuada a lo especificado (sin errores). En el apartado de diseño (interfaces) los testeadores no han puesto pegas con el diseño final.

Creo que no hay mejor indicador que los testeadores para saber si el proyecto a realizado su función. Teniendo en cuenta que es un juego, los testeadores han hecho de consumidores y han salido contentos de la experiencia. Por lo tanto es un éxito.

Como proyecto, se ha desarrollado dentro de las horas estipuladas y se ha realizado a un buen ritmo sin tener muchos retrasos ni problemas. El uso de *sprints* para sacar un producto al final de cada iteración ha sido una gran idea para evitar retrasos.

He podido confirmar la teoría de que hacer prototipos y tests mejoran mucho el producto final a pesar del tiempo que hay que invertir. En este proyecto se calcula que se ha invertido cerca del 17% (50 horas) del tiempo total en crear los prototipos y los distintos test.

Como experiencia personal, ha sido bastante bueno hacer un proyecto por mi cuenta y lanzarme a la “aventura” desarrollando algo fuera de lo común. Además aprender una herramienta desde 0 y sin ayuda creo que ha mejorado mis capacidades para enfrentarme a retos futuros que no tengan los seguros que tiene un TFG.

7.2 Lecciones aprendidas

Como lecciones aprendidas a lo largo de este TFG destacó:

- Uso del *thinking aloud* como herramienta muy útil para mejorar un producto, habría sido también util haberlo realizado en etapas cercanas al comienzo del TFG y al diseño aunque fuera con prototipos de baja calidad.

- No tener inconveniente en buscar información en otros idiomas como el inglés cuando la documentación en castellano es deficiente o nula.

- Dedicar tiempo y esfuerzo en refactorizar código antiguo o que se realizó con poca experiencia en su manejo. En este caso fue muy útil en algunos momentos releer y rehacer código antiguo cuando se descubre nuevas formas de implementación.

- Comentar y organizar el código que se está generando. Puede resultar obvio pero no siempre se hace de forma adecuada y a la larga ayuda.

7.3 Mejoras

Muchas ideas para mecánicas, mejoras visuales y otras funcionalidades tuvieron que ser descartadas ya desde que se planificó el proyecto. También ha habido funcionalidades y mecánicas que se han desechado por el camino por falta de tiempo.

- Mejorar la IA añadiendo más acciones o mejorando su capacidad de toma de decisiones.

- Mecánicas como la niebla de guerra (descartada en la segunda planificación) o la mecánica de inundar terrenos para hacerlos intransitables (descartada desde el análisis) no han podido ser implementadas.

- Mejoras gráficas y añadir efectos de sonido (aunque no sea el objetivo primordial del TFG) que mejorarán la calidad del juego.

- Más niveles y mas clases de unidades para mejorar la calidad del trabajo. En resumen usar más y mejor las herramientas que se han creado a lo largo del proyecto.

8. Bibliografía

1. admin. Si hay una industria que no es un juego, esa es la de los Videojuegos - En.Digital Podcast. In: En.Digital [Internet]. En.Digital Podcast; 6 Jun 2018 [cited 10 Apr 2019]. Available: https://en.digital/blog/videojuegos-industria-mobile-crecimiento

2. Sucasas ÁL. El crecimiento de la industria del videojuego amenaza el liderazgo del libro. In: EL PAÍS [Internet]. Ediciones EL PAÍS S.L.; 31 Jan 2019 [cited 16 May 2019]. Available: https://elpais.com/cultura/2019/01/30/actualidad/1548877027_372955.html

3. [cited 16 May 2019]. Available: https://www.google.com/url?q=http://gamestudies.org/0802/articles/sicart&sa=D&ust=1557993434778000&usg=AFQjCNExeCWKJ9a1vWSzro1xHKkvSA5-Eg

4. cited 16 May 2019]. Available: https://www.google.com/url?q=https://learn.canvas.net/courses/3/pages/level-3-dot-1-5-game-state&sa=D&ust=1557993434781000&usg=AFQjCNGegs8LedmGjjg1hrsoYlZlpxil4A

5. [cited 16 May 2019]. Available: https://www.google.com/url?q=https://en.wikipedia.org/wiki/Gameplay&sa=D&ust=1557993434780000&usg=AFQjCNHstdiq0rd-peESz6_yfByrVe2Ebg

6. Game Studies - Defining Game Mechanics [Internet]. [cited 16 May 2019]. Available: http://gamestudies.org/0802/articles/sicart

7. [cited 16 May 2019]. Available: https://www.google.com/url?q=https://es.wikipedia.org/wiki/Disonancia_ludonarrativa&sa=D&ust=1557993434783000&usg=AFQjCNFdNX-Dszl8ooIqvA8JUs0N10tXvw

8. Crawford C. THE ART OF COMPUTER GAME DESIGN. 2005.

9. Fortnite ganó 2.400 millones en 2018 y las ventas de juegos aumentan. In: HardZone [Internet]. 17 Jan 2019 [cited 16 May 2019]. Available: https://hardzone.es/2019/01/17/fortnite-ventas-juegos-2018/

10. XCOM: Enemy Unknown -. In: SteamSpy - All the data about Steam games [Internet]. [cited 16 May 2019]. Available: https://steamspy.com/app/200510

11. Heuristics [Internet]. [cited 16 May 2019]. Available: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

12. Red Blob Games: Hexagonal Grids [Internet]. [cited 16 May 2019]. Available: https://www.redblobgames.com/grids/hexagons/

13. [cited 16 May 2019]. Available: https://www.google.com/url?q=https://en.wikipedia.org/wiki/Scrum_(software_development)&sa=D&ust=1557993434784000&usg=AFQjCNGhOcVcBLzDBDNrwbYJtDf7R82alw

14. [cited 16 May 2019]. Available: https://www.google.com/url?q=http://www.gamasutra.com/view/feature/168647/making_lean_startup_tactics_work_.php&sa=D&ust=1557993434784000&usg=AFQjCNFrnoND-

VdB2jSBHRt77T9SC8UMHg

15. LaCalle A. Pensando en alto: Thinking Aloud [Internet]. [cited 18 Jun 2019]. Available: http://albertolacalle.com/hci/thinking-aloud.htm

16. Stencyl: Make iPhone, iPad, Android & Flash Games without code [Internet]. [cited 16 Jun 2019]. Available: http://www.stencyl.com/

17. CRYENGINE. In: //www.cryengine.com [Internet]. [cited 16 Jun 2019]. Available: https://www.cryengine.com/showcase

18. Contributors to Wikimedia projects. To the Moon - Wikipedia, la enciclopedia libre. In: Wikimedia Foundation, Inc. [Internet]. 24 Sep 2013 [cited 16 Jun 2019]. Available: https://es.wikipedia.org/wiki/To_the_Moon